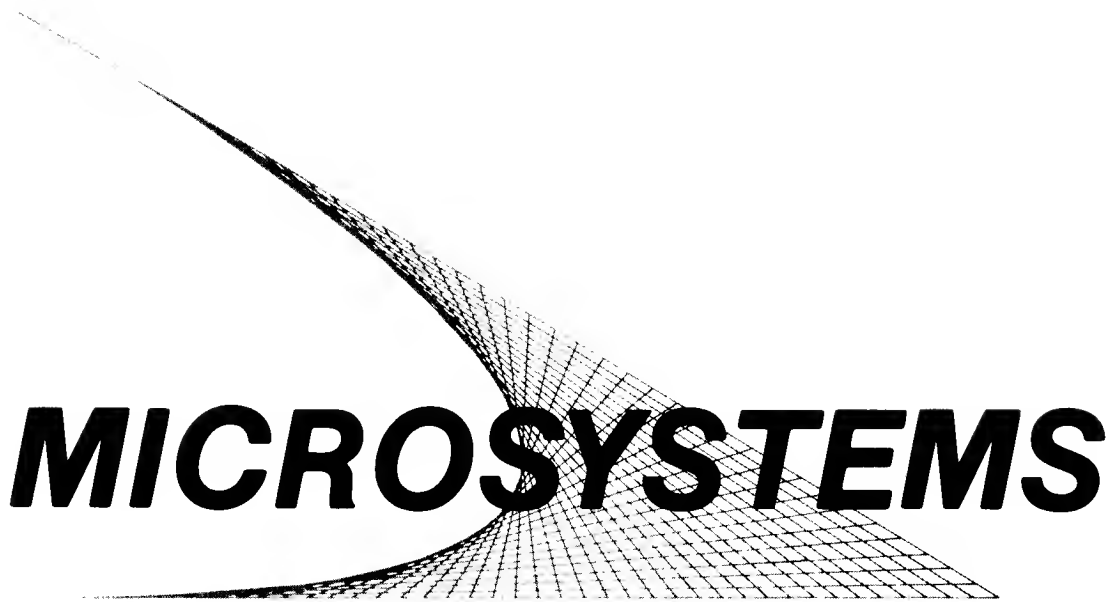
MOTOROLA

M68KLINK/D7

M68000 Family
Linkage Editor
User's Manual



QUALITY • PEOPLE • PERFORMANCE

MOTOROLA

M68KLINK/D7

JANUARY 1986

M68000 FAMILY

LINKAGE EDITOR

USER'S MANUAL

Seventh Edition

Sixth Edition March 1985

MICROSYSTEMS

REVISION RECORD

M68KLINK/D6 -- March 1985. Reflects the following software levels: VERSAdos 4.4 and Link 1.8. Adds support of the MC68020, VM04, MVME120, MVME121, MVME122, and MVME123. New options: The ATTRIBUTE interactive user command now accepts the new attribute R, which designates a real-time task, and a new interactive user command of PAGESIZE that enables modification of the page size of the load module.

M68KLINK/D7 -- January 1986. Adds support of the MVME117 and MVME130/131. Makes minor corrections to the manual and adds a keyword index.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 SCOPE

This manual explains the system features of the M68000 Family Linkage Editor (referred to also as the "linkage editor" or simply the "linker"), and describes the operating procedures necessary to link high-level languages (such as Pascal) and/or assembly language routines to create relocatable object modules, transportable S-record modules, or absolute load modules for use on M68000 Family systems. Appendices to this manual provide additional details of the three types of output, as well as examples.

1.2 INTRODUCTION

The linker examines and gathers information from the relocatable object module(s) associated with independently compiled or assembled source code module(s) and, based on this information, allocates memory to code and data, relocates according to this allocation, and resolves all references to symbols assumed to be global to one or more modules.

1.3 OPERATING ENVIRONMENT

- VERSAdos Operating System

1.4 HARDWARE REQUIREMENTS

Any of the following:

- EXORmacs Development System
- VMC 68/2 Microcomputer System
- VME/10 Microcomputer Development System
- VMEsystem
- VERSAmodule 01, 02, 03, or 04 Monoboard Microcomputer-based system
- MVME101, MVME110, MVME117, MVME120/121, MVME122/123, or MVME130/131 VMEmodule Monoboard Microcomputer-based system

1.5 THEORY OF OPERATION

Relocatable object modules are created by processing programs written for the M68000 Family microcomputers by a compiler or assembler. An absolute binary load module (or simply a "load module") is created when the object modules are relocated and linked by the linker. Later, the load module is loaded into memory through VERSAdos and executed on any of the above systems. A load module requires no further alteration.

Instead of creating a load module, the linker may optionally create a relocatable object module, combining all its input. A group of completely debugged, interrelated object modules are combined for this module. Under certain circumstances, this may be desirable, because they can be referenced by a single filename. The linker also operates faster because references between the modules are resolved when the modules are combined.

A third type of linker output is also available: an S-record format module. Data in S-record format is ASCII character data (refer to Appendix C), that facilitates the transfer of these files between computer systems. When a firmware debug monitor, such as MACSbug, TENbug, or VME110bug, is running, the **LO** command is available to "download" the S-record format files directly into memory. The **LO** command converts the S-record output modules into absolute load modules and places the load modules directly into memory. VERSAdos has a utility, **MBLM**, that converts S-record modules to load modules that can run on VERSAdos.

The linker requires two passes (the input is read twice) before it can create an output module. During the first pass, the linker gathers information about externally referenced and externally defined symbols, building a symbol table in the process. It also keeps track of the sections assigned, their names, lengths, and starting addresses. Finally, the linker determines the modules required from the library file(s). During pass one, the linker pays no attention to the actual code/data in the inputted relocatable object modules.

After pass one, if an S-record module or absolute load module is being generated, the linker assigns each section to an absolute address in memory. When executing the absolute load module, the section is loaded at this address. The allocation of memory is a complex task that can be left totally up to the linker, or influenced by various user commands (refer to Chapter 4).

After allocation of memory, the required space is known for the resulting load module or S-record module and the output file is allocated. If a relocatable object module is being generated instead, the linker computes the total size of each section in use; opens the output file and outputs the necessary information about each section and symbol to it. (NOTE: Each section always starts on a word boundary.)

The linker then proceeds to pass two, where the relocatable object modules read in pass one are re-read in the same order. Now the data/code in each module is relocated, reference resolution is done, and the data/code is written to the output file. However, if a relocatable object module is being produced, the input is not relocated, but references between the input modules are resolved.

At the completion of pass two, the linker outputs its final listings. The listings produced are based on the options specified in the invoking command line.

1.5.1 Sections

The basic unit of input to the linker is a relocatable object module. An object module consists of code/data, along with information describing externally defined symbols, externally referenced symbols, a command line address, an entry point address, and sections (the logical units where code/data are placed).

Each object module may contain code/data to be placed in up to 16 relocatable sections, numbered 0-15. Each of these sections may also have a variable number of named common sections associated with it, each containing a combination of code and data but not data only, and with each common section having a unique name (refer to Chapter 2).

In addition, an object module may contain a variable number of absolute sections. An absolute section is absolute only because an address in memory has already been assigned where it will reside at execution time. Absolute sections may contain code/data that the linker is required to modify and are not numbered.

Each relocatable section is designated as either a "regular section" or a "short section". A regular section is a relocatable section that may be placed anywhere in memory for execution. Thus, wherever an absolute reference is made to something in a regular section, a full long word (4 bytes) is reserved for the absolute address, because there is no guarantee where the linker will place the regular section in the load module.

A short section is a relocatable section that should be located in memory addressable by a short absolute address (a 2-byte address). Whenever an absolute reference is made to something in a short section, only one word is reserved for the address. This word can address the first 32Kb and the last 32Kb of memory. (NOTE: Some target processors provide for an upper limit on memory of $FFFFFF, $FFFFFFF, or $FFFFFFFF). By putting frequently-used code/data in short sections and locating these sections properly, a program can be shorter, because each reference to that code/data will require a 2-byte, rather than a 4-byte, address. A single short section may be located completely within the lower or upper absolute short address range, but may not occupy both.

1.5.2 Segments

A segment is the basic unit of memory where sections may be placed. It is defined by its name, starting address, and length. Its length must be a multiple of 256 bytes to be loaded by VERSAdos. When the system contains a Memory Management Unit (MMU), the starting address is a logical address that is translated by the MMU into a physical address in memory. On a non-MMU system, the starting address represents a physical address in memory and VERSAdos attempts to load the program at the specified physical address. However, if the first segment of a program starts at address 0, the program is assumed to be position independent and VERSAdos loads it into the next available memory space.

A load module may contain up to four segments, each containing code/data. When loading the module into memory, a non-MMU system verifies that the specified segments of physical memory are available and loads them if possible (unless program is position independent). For systems that contain a MMU, a segment may be placed anywhere in available memory that is large enough to hold it. The MMU maps all logical addresses generated by the user code/data into the physical addresses in memory where the segment resides. For example, a segment may logically start at hexadecimal address $1E00, with a length of 1024 bytes. However, the segment may actually reside in physical memory between addresses $2F600 and $2F9FF, inclusive. Then, if code in the user program requests data at address $1F76, the MMU maps the address into physical address $2F776 and retrieves the data from there.

When creating a binary load module, the linker sets up the desired segments and assigns each section being used to a segment. The definition of segments and what sections they contain may either be left totally up to the linker or influenced by the user. Refer to paragraph 4.18 for details.

When the linker's output is an S-record module, the linker allocates segments as above, but the segment information is not carried over when downloading. However, segments can be assigned when using the VERSAdos utility, **MBLM**, to convert the S-record files to an executable load module.

1.5.3 Relocation and Linking

The input to the linker is relocatable object modules produced by the linker itself, the assembler, or a compiler such as Pascal. The term "relocatable" means that the data in the module has not been assigned to absolute addresses in memory; instead, each different section is assembled/compiled as though it started at relative address 0. (The exception to this is absolute sections, which do get assigned to absolute addresses at assembly time.) When creating an absolute load module or S-record module, it is the job of the linker to read in all the relocatable object modules that comprise a program and assign each section to an absolute memory address. Then, in the process of actually putting the code and data read from each object module into the proper location in the load module, the linker must fill in the correct addresses for such items as loading absolute addresses and referencing across sections. This is the process of relocation.

NOTE

There is a difference between relocatable and position-independent code. A program is "position-independent" if it runs correctly when the exact same machine code is positioned arbitrarily in memory. The output of the Pascal compiler is both relocatable and position-independent. The output of the assembler is relocatable, however, it is up to the user to produce position-independent code, if desired. If the input to the linker is position-independent, the load module produced by the linker is also position-independent, even though it has been assigned to absolute memory addresses.

Along with relocation, the linker performs reference resolution between modules, i.e., one module can reference symbols defined in a different module. This is done in the following manner:

> At the time of compilation/assembly, the module doing the referencing has no idea where the symbol it is referencing will be located in the final load module. Therefore, the compiler/assembler sets up information in the relocatable object module that indicates an external symbol is referenced in this module and where it is referenced. This is an "**XREF**". Also, in the relocatable object module, where the symbol is defined, there is information that indicates the reference, along with the relative address of the symbol in the module. This is an "**XDEF**". The correct address of the symbol is inserted wherever it is referenced, when the two modules are input to the linker.

Using relocation and linking allows the user to break up a large program into separate, more manageable modules that may be assembled or compiled separately. These modules may then be linkage edited to produce a load module or S-record module of the complete program. If a problem is encountered, only the module with the problem needs editing and re-compiling/re-assembling. Then the user can do reference resolution between the new relocatable object module and the other previously created object modules to create a new load module.

1.5.4 Library Files

A library file contains several relocatable object modules, each of which contains one or more external symbol definitions (**XDEF**s). A library file is not "input" to the linker the same as regular relocatable object modules, instead, a library file is "searched". A search of a library file consists of looking one-by-one at each relocatable object module in the file.

For each module, a check is done to determine if any externally defined symbols in the module match any unmatched externally referenced symbols (**XREF**s) from other modules input. If so, the relocatable object module is included in the load module. If not, the module is not included and the search continues with the next relocatable object module in the library file. A search of a library file continues until the end of the file is encountered or all unresolved external references are resolved, whichever occurs first.

A library file is created by separately assembling or compiling each module that is to go in the library. Each module contains definitions (**XDEF**s) for the proper external symbol(s). The resulting separate relocatable object modules are merged into one file using the **ADD** subcommand of the **LIB** utility under VERSAdos.

NOTE

> The linker cannot create a library file, since its output is a file that contains only one relocatable object module containing the data from all the input modules.

1

1.6 RELATED PUBLICATIONS

The following publications may provide additional helpful information. If not shipped with this product, they may be obtained from Motorola's Literature Distribution Center, 616 West 24th Street, Tempe, AZ 85282; telephone (602) 994-6561.

DOCUMENT TITLE	MOTOROLA PUBLICATION NUMBER
M68000 Family Resident Structured Assembler User's Manual	M68KMASM
M68000 Family Resident Pascal User's Manual	M68KPASC
M68000 Family Resident FORTRAN Compiler User's Manual	M68KFORTRN
VERSAdos Messages Reference Manual	M68KVMSG
VERSAdos System Facilities Reference Manual	M68KVSF
VERSAdos Data Management Services and Program Loader User's Manual	RMS68KIO
M68000 Family Real-Time Multitasking Software User's Manual	M68KRMS68K

1.7 NOTATION

The following conventions are used in the command syntax, examples, and text in this manual:

boldface string — A boldface string is a literal such as a command or a program name, and is to typed just as it appears.

< > — Angular brackets enclose a "syntactic variable", to be replaced by one of a class of items it represents.

| — A vertical bar separating two or more items indicates that a choice is to be made; only one of the items separated by this symbol should be selected.

[ ] — Square brackets enclose an item that is optional. The enclosed item may occur zero or one time.

[ ]... — Square brackets followed by periods enclose an item that is optional/repetitive. The item may appear zero or more times.

A carriage return follows all operator entries. If the only input required is a carriage return, it is shown as (**CR**).

CHAPTER 2

MEMORY ALLOCATION

2.1 GENERAL

This chapter discusses memory allocation. For absolute load modules and S-record modules, memory allocation performed by the linker is essentially the same. Paragraph 2.2 is generally applicable to both; however, refer to paragraph 2.3 for additional information about S-records. Paragraph 2.4 covers the linker's handling of relocatable object modules.

2.2 WHEN CREATING A LOAD MODULE

Allocation of memory is a complex process and may, in many instances, be left totally up to the linker. However, the user may slightly or greatly affect the allocation of memory if the application so warrants, with **SEGMENT** and **START** user commands (refer to paragraphs 4.18 and 4.19). Before using these commands, the user should be familiar with the process the linker goes through in the allocation of memory.

At the end of pass one, after all the relocatable object modules that will create the load module have been read once, allocation of memory takes place. Now it is known what sections, and their sizes, will be used to create the output module. The segments where these sections will be placed is also known. This is determined by the **SEGMENT** commands the user supplied; or, if **SEGMENT** commands were not given, the following defaults take effect:

a. SEG0 will contain sections 0 through 7, as a Read/Write segment.
b. SEG1 will contain sections 8 through 14, as a Read only segment.
c. SEG2 will contain section 15, as a Read/Write segment.

The process of memory allocation begins with the allocation of absolute sections. This consists of going through all the absolute sections encountered in the relocatable object modules input, and reserving the memory defined by each section.

The linker then proceeds to allocate each segment. There are two basic types of segments, and segments are allocated in order of type. Within each type, segments are allocated in the order they were introduced to the linker. The two types are:

Type 1 Segments with starting or starting and ending addresses specified (via a **SEGMENT** command).

Type 2 Segments with no starting or ending addresses specified (via a **SEGMENT** command or if defaults taken).

A segment allocation is determined by the starting address of the segment, allocating the sections in the segment, determining the ending address of the segment, and ensuring that all the memory between the starting and ending addresses is reserved. If the segment is type 1, the starting address of the segment is the starting address the user specified. Otherwise, the starting address chosen for the segment depends on the value of the **S** option (refer to paragraph 3.4). If the **S** option is off, the linker scans from low memory to high memory, looking for the first available block in which the segment will fit, and uses the starting address of that block. This is the "first fit" approach. However, if the **S** option is on, the linker starts the segment immediately after the segment with the highest starting address amongst those segments that are already allocated. If segments have not been allocated, the segment starts at address 0.

Once the starting address of the segment is determined, the segment is allocated section by section. There are two basic types of sections: those that have been assigned where to start (via a **START** command) and those that have not. The sections assigned starting addresses with **START** commands are allocated first, in order of increasing starting address. If a list of sections was specified in a **START** command, those sections are allocated one right after another in the order specified, starting at the address given. After allocating the sections that were given starting addresses, the sections without starting addresses are allocated. These sections are allocated in the order given in the **SEGMENT** command (or in the order of increasing section number, if no **SEGMENT** commands were given). Space for these sections will be looked for starting at the beginning of the segment the sections are in. Once enough space is found for the first section, it is allocated. Then space for the next section will be looked for and from that point on, until all the sections are allocated. (NOTE: This does not require that all sections without starting addresses be allocated in one contiguous block.)

When a section is allocated, all the sections of that number encountered in the inputted relocatable object modules are placed contiguously (each module on the next word boundary), in the output module. They are placed in the order in which they were initially read. Furthermore, if the section has any common sections associated with it, they are placed contiguously in the load module immediately following, again in the order they were initially encountered. Thus, unless the **B** option is on, all sections of the same number, including common sections, are always allocated in one contiguous block of memory.

An exception to this method of allocating sections is when the **B** option is used (refer to paragraph 3.4). When the **B** option is on, each relocatable section from each object module is forced to start on a page (256 byte) boundary. Thus, all sections of the same number are not necessarily allocated contiguously, one right after another. "Holes" are left between the end of one section and the start of the next page. Also, those sections that have a designated starting address via **START** commands, start on the first page boundary after the address specified in the **START** command (unless that address is already a page boundary).

The ending address of the segment is determined following the allocation of sections within a segment. If the user specified an ending address, then it is used. Otherwise, the highest ending address of all the sections in the segment is used.

Finally, the linker ensures that all memory between the starting and ending addresses of the segment just allocated is flagged as being used.

There is a special case of allocating a segment, i.e., if sections that go in the segment are not found in the inputted relocatable object modules. In other words, the segment is essentially empty and one of three actions will take place:

a. If the user specified a starting and ending address for the segment, the segment will be allocated using those addresses.

b. If the user only specified a starting address for the segment, the segment will be allocated and will be 256 bytes long.

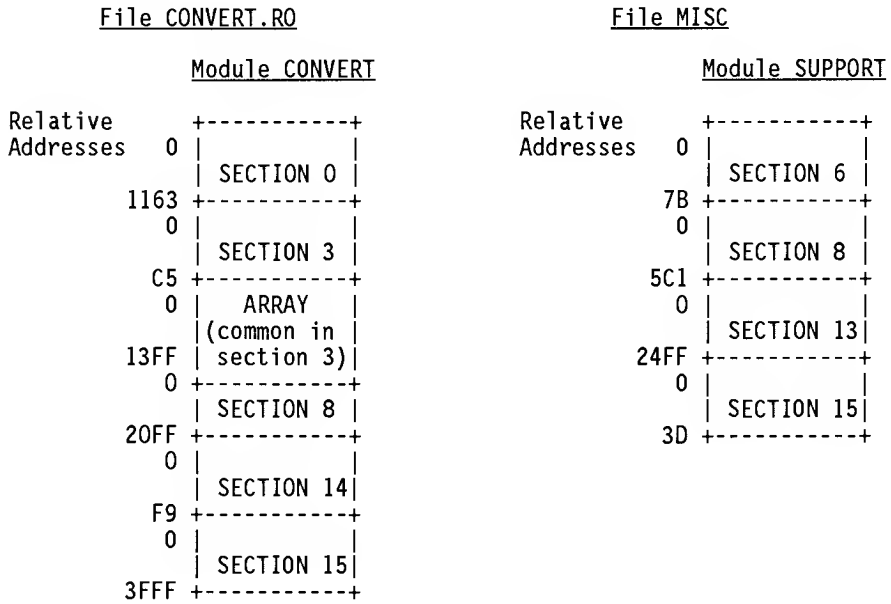
c. If no starting or ending address was specified for the segment, it will not be allocated and will not appear in the load module.

After all segments have been allocated observing the above rules, one final action is taken to ensure that all absolute sections are entirely contained within segments. This is done by looking at each absolute section and determining if it is already entirely contained within a segment. If so, the linker proceeds to the next absolute section. If not, the linker attempts to get the absolute section entirely within a segment by modifying either the ending address of the segment immediately before the section in the load module or the starting address of the segment immediately after the section. Only those addresses not supplied by the user will be modified. In other words, if the user supplied the starting, or starting and ending, addresses of a segment, those addresses are never changed.

If no segments are allocated (refer to c. above), when it is time to check the absolute sections, segment 0 will be designated as read/write and set up to contain all the absolute sections. It will start at the beginning address of the first absolute section and will be just as long as necessary to contain all the absolute sections.

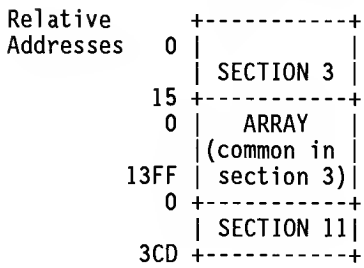
2.2.1 Example 1

This example illustrates a simple memory allocation using the defaults supplied by the linker. The example is in three parts, the first of which is an illustration of the files and modules input to the linker. Next is the command line used to invoke the linker, and the last is an illustration of what the structure of the resulting load module will be.

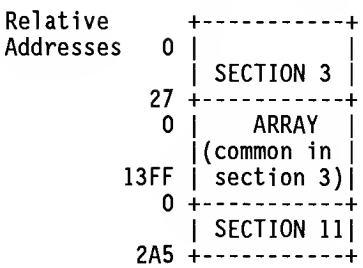


File CONVSUBS

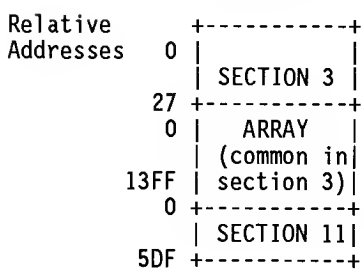
Module INVERT



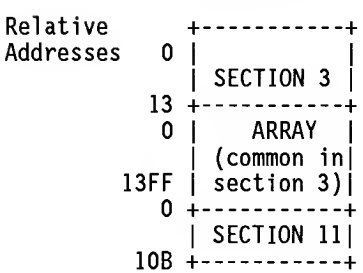
Module MULTIPLY



Module PRINT



Module MOVE

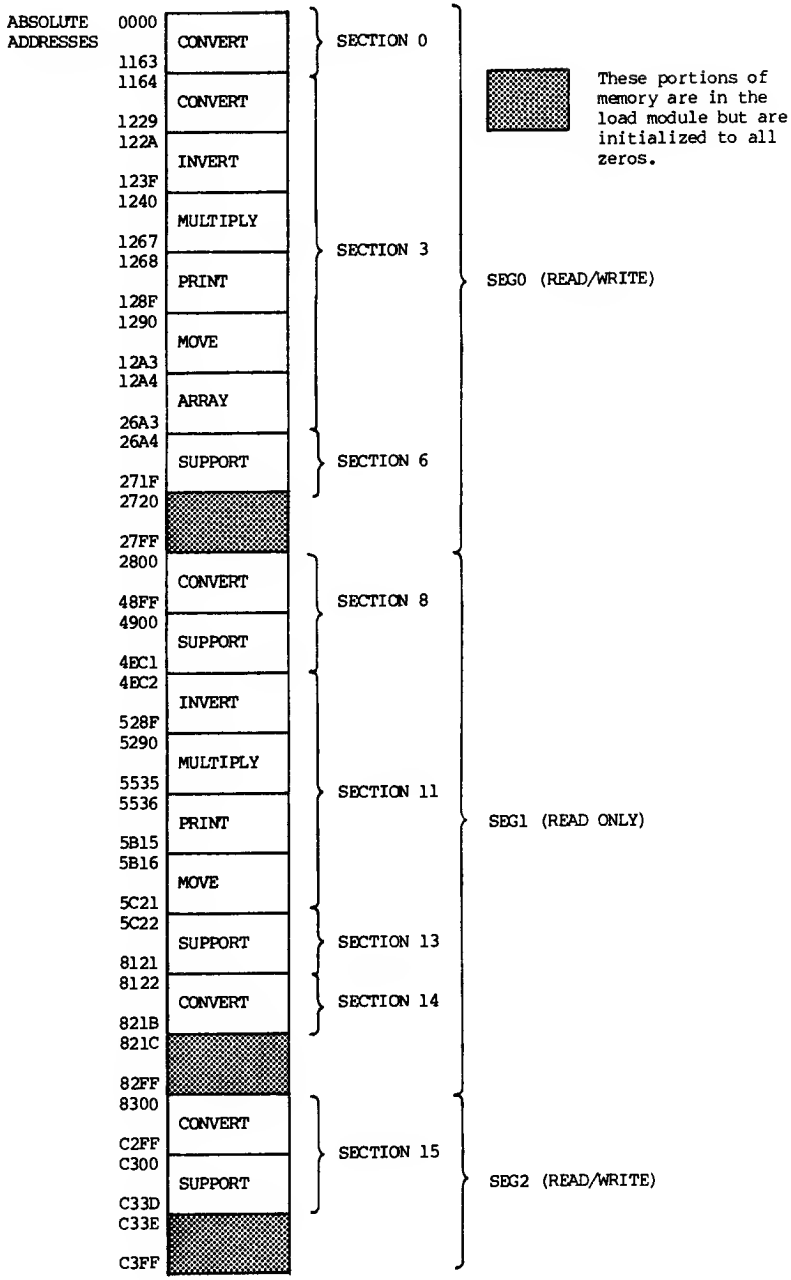


Invoking Command Line

```
=LINK    CONVERT/CONVSUBS/MISC
```

2

RESULTING LOAD MODULE

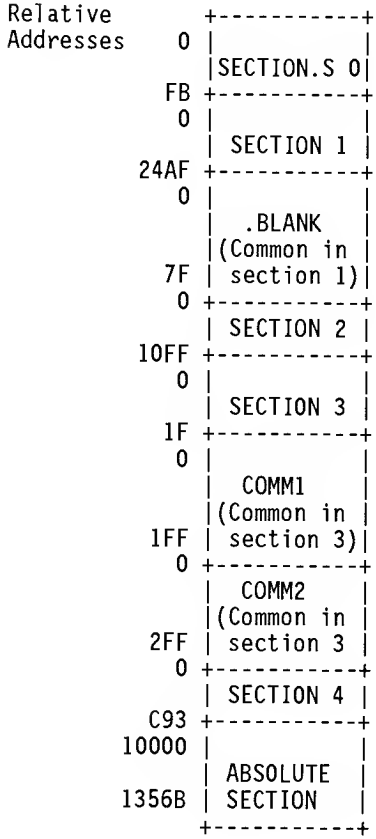


2.2.2 Example 2

This example applies user commands to generate a more complex memory allocation. The first part illustrates the files input to the linker, the relocatable object modules in the files, and the sections comprising the modules. Next is a list of user commands given to the linker to process those files. Last is an illustration of the resulting load module. Note that this example corresponds to the first example listing given in paragraph 5.3.

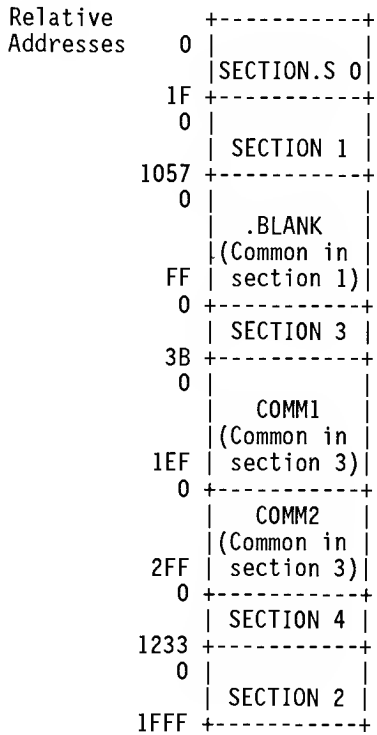
File MAIN.RO

Module MAIN



File SUBRTS.RO

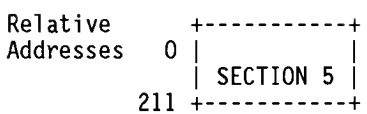
Module SPROGS



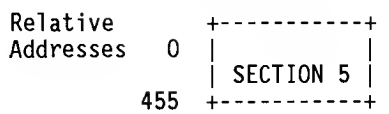
2

File MATH.RO

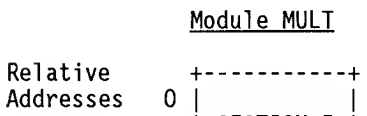
Module ADD



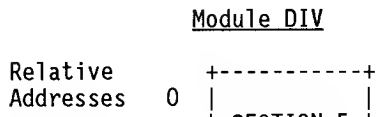
Module SUB



Module MULT

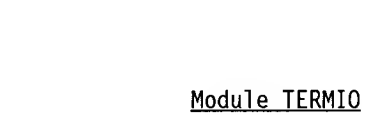


Module DIV

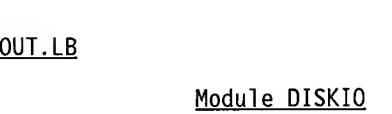


File INOUT.LB

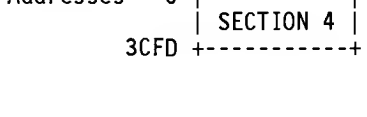
Module TERMIO



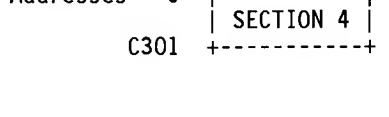
Module DISKIO



Module PRINTIO



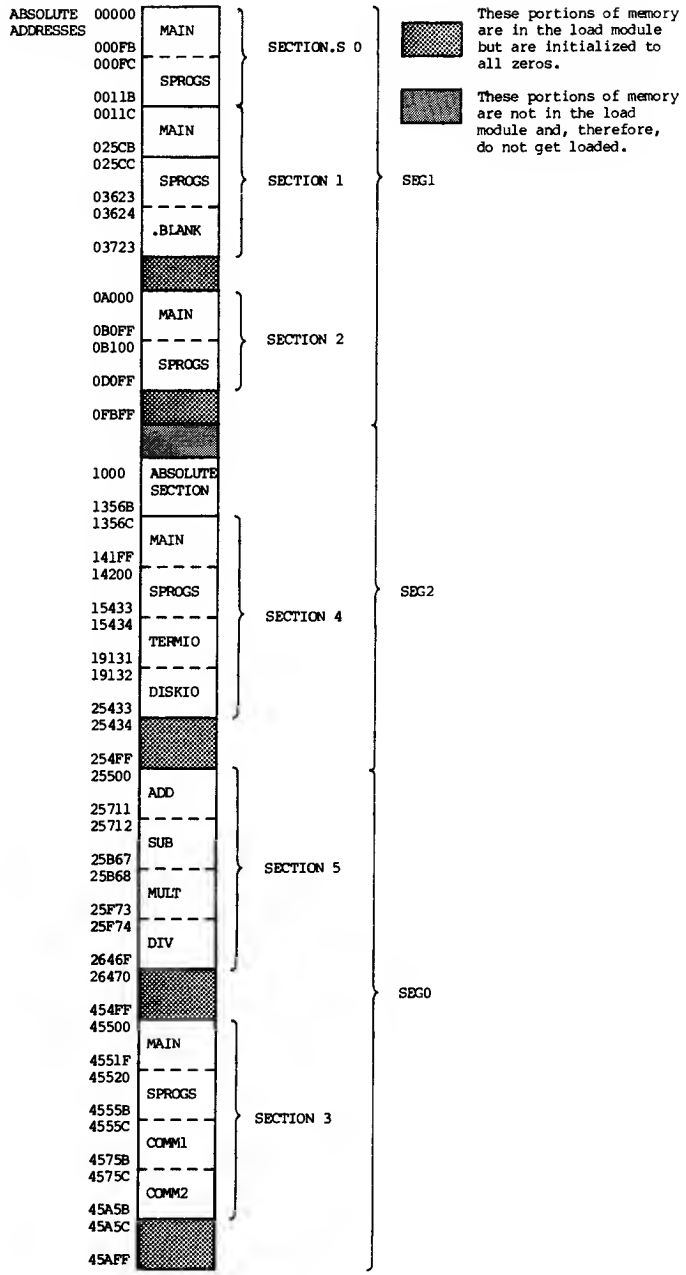
Module TAPEIO



User Commands

```
SEGMENT SEG1(RL):0-2  0,$FBFF
SEGMENT SEG0:5,3
SEGMENT SEG2:4 64K
START   2 $A000
START   3 $20000
INPUT   MAIN, SUBRTS
INPUT   MATH<ADD,SUB,MULT,DIV>
LIB     INOUT.LB
END
```

RESULTING LOAD MODULE



2.3 WHEN CREATING AN S-RECORD MODULE

When the output of the linker is an S-record format module, the linker handles the memory allocation similar to a load module (refer to paragraph 2.2). However, when the module is downloaded to a target system, the segment allocation descriptors are not carried over. In the VERSAdos case, when the downloaded S-record module is converted to an executable load module using the utility **MBLM**, the segment names, locations, and lengths can be re-specified by the user, after referring to the printout of the load map produced by the linker.

2.4 WHEN CREATING A RELOCATABLE OBJECT MODULE

When the linker creates a relocatable object module, memory is not actually allocated, although the following modification/re-arrangement of the relocatable object module takes place:

a. All relocatable sections of the same number are combined into one section with that number.

   All common sections of the same number/name are combined into one common section, and its size is determined by the largest size input for that section.

   The absolute sections encountered in the input, however, are passed directly to the output module. Even if two absolute sections are contiguous with each other, they will not be combined into one larger section, but passed on as two separate absolute sections.

b. If a reference is made to an external symbol that is defined in one of the object modules input, that reference will be resolved and its value put into the resulting object module. Otherwise, the code and data in the relocatable object modules input is unchanged in the module being created.

   All the external symbol definitions and unresolved external symbol references encountered in the input are grouped together in the output module. However, if any **XDEF** commands are used (refer to paragraph 4.21), only those externally defined symbols in the **XDEF** command(s) will be externally defined symbols in the output module.

CHAPTER 3

INVOKING THE LINKER

3.1 GENERAL

The linker is a disk-resident program loaded into memory by VERSAdos in response to a **LINK** command. This **LINK** command line may include certain parameters and options. Alternatively, certain parameters and options may be specified in user commands (refer to Chapter 4).

3.2 FILENAME FORMAT

"Filename" refers to a directory entry for a physical module on a disk or diskette. It includes six descriptor fields (refer to the VERSAdos System Facilities Reference Manual). One field is also called "filename", and may be (and frequently is) used alone to specify a directory entry. Other descriptor field combinations are also allowed.

The filename combinations acceptable to the linker are:

<filename>

<filename>.<ext>[(<prot>)]

<catalog>.<filename>.<ext>[(<prot>)]

<user#>.[<catalog>].<filename>[.<ext>[(<prot>)]]

<voln>:[<user#>].[<catalog>].<filename>[.<ext>[(<prot>)]]

3.3 COMMAND LINE FORMAT

VERSAdos will bring the linker into memory and begin its execution in response to a **LINK** command. Parameter or option information given with the **LINK** command line is saved for use by the linker. The format of the command line is:

=LINK [<fn1>[/<fn1>]...][,[<fn2>][,<fn3>|#|#PR|#PRn]][;<options>]

where the syntactic variables are defined as:

<fn1> Input file(s). These are names of disk files (one or more), each containing one or more relocatable object modules. As many input files as desired may be specified on the command line. Extensions, if not given, default to .RO. These files are processed before any files specified by **INPUT** commands.

If input files are not specified, the **A** option (refer to paragraph 3.4) is forced on to allow user commands. Files are then specified using the **INPUT** command (refer to paragraph 4.11).

<fn2> Output file. The name of the disk file that will contain the output of the linker. This file will contain a load module, a relocatable object module, or an S-record module (depending on the options used). If a load module or an S-record module is being created, the filename need not be specified, because the linker assumes the name of the first input file processed with an extension of .LO or .MX. If a relocatable object module is being created, an output filename that is different from the input filename(s), must be specified. The default extension for this filename is .LO, .RO, or .MX, depending on whether a load module, relocatable object module, or S-record module is being produced.

<fn3> Listing file. This is the name of the disk file, with default extension of .LL, that will contain the listings produced by the linker.

**#PR** **#PRn**

If **#**, **#PR**, or **#PRn** is specified instead of <fn3>, the listings will be directed to the user's console or line printer, respectively.

If no listing file/device is specified, but options requesting listings are, the listing will be directed to the default output file/device (usually the user's console).

<options> This is one or more of the options described in paragraph 3.4.

3.4 OPTIONS

The linkage editor has two types of options: a single letter (preceded by a minus sign if the option is to be disabled), or a letter followed by an equal sign and a related field. When multiple options are specified, options of the first type may be separated by a comma, or have no separation; options of the second type must be separated from following options by commas. The options may be specified in any order; the first option specified must be preceded by a semicolon.

A (Default: **-A**) Accept user commands from the command input device. If input files are not specified in the command line, this option is forced on.

B (Default: **-B**) In the listing produced by the assembler, each relocatable section in a module starts at relative address zero. However, each actual starting address (offset) is wherever the linker locates a section within a memory segment. Therefore, to form actual addresses for a section, this offset must be added to each relative address in the listing.

To assist in the process, the **LINK** command line accepts a **B** option, which forces each relocatable section from each module to start on a page ($100-byte) boundary. The offset then appears as $xxxx00, which, being a multiple of $100, makes it easier to work with and remember.

This option, however, does not affect an absolute section, which is always placed at the address indicated by its **ORG** directive.

If a **START** user command, following a **B** option, defines a starting address that is not on a page boundary, the particular section or sections will start at the first page boundary after that address.

This option may be used only if a load module or an S-record file is being created.

NOTE

In linking a program that consists of many individual sections from many modules, the **B** option could greatly increase the size of the resulting load module.

The **B** option should be a tool that is used for debugging purposes only. Once a stable, debugged program is achieved, it should be re-linked without the **B** option to produce a final load module.

D (Default: **-D**) Create a debug file. If this option is specified, a file will be produced containing information pertinent to symbolic debugging. It will have the same name as the first file processed for input, with an extension of .DB. Note that this option and the **R** option are mutually exclusive.

H (Default: **-H**) List information found in the header record of each object module input on the listing file. Refer to printout format #5 in Chapter 5.

I (Default: **-I**) List the command line and all user commands, if any, on the listing file. Refer to printout formats #2 and #4 in Chapter 5.

L=<fn>[/<fn>]...

(Default: **-L**) Search the specified library files in the order listed, if any references are unresolved at the end of pass 1. Process any modules that contain definitions satisfying any unresolved references. A library is searched only once; any modules that reference other modules within a library file should occur within the library file before the referenced modules.

This option must be followed by a comma unless it is the last option in the command line.

M (Default: **-M**) List a map of the resulting module on the listing file. Refer to printout format #6 in Chapter 5.

O (Default: **O**) Create an absolute binary load module. Specifying this option combines the inputted relocatable object modules and creates an absolute load module. If no output filename was specified in the command line, the load module will have the same name as the first file processed for input, but with an .LO extension. Note that this option and the **R** and **Q** options are mutually exclusive.

P (Default: **P** or **-P**) Search default libraries at the end of pass 1 if unresolved external references. There is one default library file supplied for each language supported by VERSAdos, for example, 0.&.FORTLIB.RO or 0.&.PASCALIB.RO. The libraries to be searched and the order in which they are searched will be determined by what language processors were used to create the object modules input and the order in which the object modules were processed.

This option defaults to on (**P**) if a load module or an S-record module is being created (**O** or **Q** option is on). Otherwise, it defaults to off (**-P**).

The **L** option, if specified, is executed first, to load any user-written modules before default library modules.

Q (Default: **-Q**) Create an S-record output module. If the output filename is not specified, it defaults to the name of the first input file, plus the .MX extension. When **Q** is specified, the user commands **TASK**, **MONITOR**, **PRIORITIES**, **OPTIONS**, **ATTRIBUTES**, and **COMLINE** may not be used, but the **IDENT** command may be used to specify identification to the S0 record. Note that this option and the **O** and **R** options are mutually exclusive.

R (Default: **-R**) Create a relocatable object module. This option requests that the relocatable object modules input be combined to create another relocatable object module, rather than an S-record module or an absolute load module. All references between the modules input will be resolved. Only those external references that cannot be resolved among the input modules will be included in the output module. All the external symbol definitions encountered in the input modules will be included in the output module unless an **XDEF** user command is specified (refer to paragraph 4.21). When the **R** option is used, an output filename, different from the input filename(s), must be specified on the command line; otherwise, an error results. Note that this option and the **D**, **O**, and **Q** options are mutually exclusive.

S (Default: **-S**) When the **S** option is used on the **LINK** command line, segments without user-specified starting addresses are allocated sequentially, on page boundaries, after the segment having a user-specified starting address. Allocation occurs in the order segments are defined in **SEGMENT** commands.

For example, when only one segment is given a user-specified starting address, that segment will be allocated at that address, and all remaining segments will be allocated immediately after it.

When more than one user-specified starting address is given, segments without user-specified starting addresses are allocated after the segment having the highest user-specified starting address.

If user-specified starting addresses are not specified, the **S** option has no effect. The segments are allocated sequentially, starting at memory address 0.

When **-S** is in effect, segments without user-specified starting addresses are allocated sequentially in the order they are encountered by the linker, on a "first-fit" basis.

Use this option only if a load module or an S-record module is being created.

U (Default: **-U**) If any unresolved references exist at the end of pass 1, list the references. Allow the user to specify additional commands to resolve the references. This option is forced off if the command input device is not the user console.

IMPORTANT: If this option is specified, all unresolved references must be resolved before the linker proceeds to pass 2.

W=<number>

(Default: **W=24**) Valid <number> may be 24, 28, or 32. Specify the bit width of the addressable memory space. Some target processors may provide for a 28-bit or 32-bit memory space (e.g., the MC68010 or the MC68020). Use of the **W=28** or **W=32** option allows the user to create an S-record output module that contains 28-bit or 32-bit addresses. **W=28** or **W=32** may only be used when the **Q** option is on.

X (Default: **-X**) List the external definition directory on the listing file. Refer to printout format #7 in Chapter 5.

Z=<number>

(Default: **Z=35**, which allocates 35,840 bytes) Allocate a stack and heap segment of at least <number>Kb (1Kb = 1024). This segment is used by the linker for storage of the symbol table. If the linker aborts with a Pascal runtime abort code of $1008, $1010, or $1011 (refer to VERSAdos Messages Reference Manual or M68000 Family Resident Pascal User's Manual), it may be possible to do the link successfully by invoking it with a larger **Z** option.

3.5 EXAMPLES

Sample command lines:

```
=LINK MAIN/SUB1/SUB2/SUB3,,#PR;HMA
```

This command line causes the linker to read for input the files MAIN.RO, SUB1.RO, SUB2.RO, and SUB3.RO. The user will then be prompted for user commands. The resulting load module will be named MAIN.LO. The final listings, including those requested by the **M** and **H** options, will go to the line printer.


```
=LINK VOL1:..PROGA/VOL2:..PROGA,VOL3:..PROGA.CM;L=LIB1.LB/LIB2.LB
```

This command line requests that the files VOL1:..PROGA.RO and VOL2:..PROGA.RO be used for input. After processing these two files, if there are any unresolved external references, the libraries LIB1.LB and LIB2.LB will be searched, in that order, in an attempt to resolve those references. The resulting load module will be stored on volume VOL3 under the name of PROGA.CM. The final listings will be output on the default output device.

```
=LINK ,OUTPUT,#;RZ=60
```

This command line indicates that the resulting relocatable object module be put into a file named OUTPUT.RO, and the final listings are to go to the user console. Also, a stack/heap segment of at least 60Kb (61440 bytes) should be used during the link to allow for a large symbol table. Since no input files were specified in this command, the **A** option will be forced on, which causes the user to be prompted for user commands.

```
=LINK
```

This command line simply starts up the linker. The user will then be prompted for user commands. The resulting load module will have the same name as the first file specified in an **INPUT** command, but with an extension of .LO. The final listings will be printed on the default output device.

CHAPTER 4

USER COMMANDS

4.1 GENERAL

For some program applications, all the linker parameters may be entered in the **LINK** command line. However, user commands (refer to Table 4-1) provide alternate and additional forms of input for certain parameters and options.

TABLE 4-1. User Commands

TASK DEFINITION	INPUT	RELOCATABLE OUTPUT	LISTING	MEMORY	CONTROL
ATTRIBUTES	DEFINE	IDENT	LIST	COMLINE	ABORT
MONITOR	INPUT	XDEF	LISTM	ENTRY	END
OPTIONS	LIBRARY		LISTU	PAGESIZE	QUIT
PRIORITIES			LISTX	SEGMENT	
TASK				START	

The linker requests user commands when any the following conditions exist:

a. Input files are not specified in the command line.

b. The **A** option is on.

c. Unresolved external references exist at the end of pass 1, and the **U** option is on (the user's console must be the command input device).

The linker prompt, a right angle bracket (>), is output at the beginning of the line and the linker then waits for a response, i.e., a user command. The user ends each command with a carriage return. The operation requested by that command is started and, when completed, another prompt is printed. The sequence is repeated until an **END**, **QUIT**, or **ABORT** command is entered. Then the linker continues processing (refer to paragraph 4.8) or returns control to VERSAdos, respectively.

The user commands format and description are given in paragraphs 4.4 through 4.21. Although the "task definition" user commands cannot be used for S-record modules, this information can be supplied via the **MBLM** utility when the downloaded S-record files are converted to load modules.

The linkage editor allows arguments or parameter substitution with the VERSAdos session control command **ARG**. This capability (refer to the VERSAdos System Facilities Reference Manual) gives the user a shorthand notation and a way to generalize chainfiles. Argument expansion will increase the length of a user command line; the user should note that the maximum length of a user command line is 132 characters.

4.2 NUMERICAL ENTRIES

All numerical entries in user commands, except the user number in a filename, may be made in any of the following forms:

```
%<binary digits>
@<octal digits>
<decimal digits>
$<hexadecimal digits>
```

In addition, the letter **K** may follow any of the above forms. This indicates that the preceding number should be multiplied by decimal 1024 ($400). For example, the following numbers are all equivalent:

```
%101101K
@132000
45K
$B400
```

4.3 SYMBOL, MODULE, AND SEGMENT NAME FORMATS

The following describes the legal forms for specifying a symbol, module, segment, or taskname in a user command (i.e., the **SEGMENT**, **MONITOR**, **TASK**, **COMLINE**, **ENTRY**, **INPUT**, **IDENT**, and **XDEF** commands):

a. A symbol or module name may be from one to ten characters long; a segment name may be from one to four characters long.

b. The first character of a symbol or module name must be an alpha (A-Z) or a period (.). Following characters may be an alphanumeric (A-Z, 0-9), a period (.), a dollar sign ($), or an underscore (_). For a segment or taskname, the first character may additionally be an underscore or an ampersand (&), and a following character may additionally be an ampersand.

4.4 ABORT

This command, written as:

ABORT

causes an immediate, orderly halt to all processing. All open files will be closed, and control is returned to the operating system.

4.5 ATTRIBUTES

The **ATTRIBUTES** command sets up task attributes in the load module being created. Its form is:

ATTR[IBUTES] <attributes>

where:

<attributes> is a list of zero or more one-character options, which may or may not be separated by spaces.

The legal attributes and their meanings are:

S This is a system task. (NOTE: The resultant load module must be loaded from user 0 to execute as a system task.)

D Ask for a task dump if this task is aborted.

F At load time, the system loader will assign the file from which this task is loaded to this task's logical unit 8.

P This task is position independent.

R This task is real-time.

No task attributes will be present in the load module if an **ATTRIBUTES** command is given with no <attributes>, or no **ATTRIBUTES** command is specified. (Refer to Appendix G for MVME12x-specific information.)

EXAMPLES:

ATTRIBUTES S D

ATTR

ATTR DS

ATTR S

This command cannot be used when a relocatable object module or an S-record module is being created.

4.6 COMLINE

The format of the command is:

COML[INE] <name>[,<length>]|[(<seg>)]<address>[,<length>]

where:

<name> is the name of a symbol externally defined in one of the previously processed input modules.

<length> the maximum number of command line characters to be stored. When given, must be between 1 and 256, inclusive. If not specified, the default maximum command line is 160 characters. May be specified with <name> or <address>.

<seg> is a segment name and must be enclosed in parentheses.

<address> is the logical address of start of the command line. It is recognized as absolute if specified alone. If specified with segment name, it is relative to the start of that segment. Whether relative or absolute, the corresponding address must be even.

This command is used to specify where the command line that invokes the user program is to be stored by the operating system before control is passed to the user program. If the <name> parameter is used, the point at which that symbol is defined will then be the point at which the command line is stored. If a relocatable object module is being produced, this is the only way the command line address may be specified. However, if a load module is being created, the command line address may alternately be given by specifying an address. **COMLINE** is not valid when an S-record module is being created.

The user may optionally specify the maximum number of characters from the command line to be stored at the given address by specifying the <length> parameter.

If no **COMLINE** command is specified, the first command line specification encountered in the relocatable input will be used to indicate where the command line is to be stored and the maximum number of characters to be stored. If no command line specification is encountered, the output module will be set up to indicate that the command line is not to be stored anywhere.

EXAMPLES:

COMLINE (SEG3)200,80

This command indicates the command line will be stored at relative address 200 ($C8) in a segment named SEG3. At the most, 80 characters will be stored.

COML COML

The command line is to be stored at the address where the externally defined symbol COML is defined. The default value of 160 is the maximum length of the command line.

4

COML $1000,256

This command indicates that the command line is to be stored at logical address $1000. At the most, 256 characters will be stored there.

4.7 DEFINE

At link time, the **DEFINE** user command can be used to assign an absolute value to a symbol specified in an XREF directive of the program. The form is:

DEF[INE] <symbol>,<value>

where:

<symbol> is any legal symbol name that is not already specified in the program in an **XDEF** directive.

<value> is any numerical value between $0 and $FFFFFFFF, inclusive.

A symbol defined this way will satisfy any reference to a symbol of the same name.

When a relocatable object module is being created, any symbols specified in a **DEFINE** command are carried along in the external symbol definition table of the new module. However, if an **XDEF** user command is given, only those symbols specified in the command will be carried along in the new module.

EXAMPLES:

```
DEFINE PHRED,$3456FF
DEF SYM1,@177723
```

4.8 END

This command, written as:

END

signals the end of user commands, and must be the last command given. A number of events occur when the **END** command is encountered:

A check is made to determine if there are any unresolved external references. If there are, the libraries specified in the **L** option, if any, are searched in an attempt to resolve the references. If unresolved references still exist and the **P** option was specified, the default libraries are also searched. If there are still unresolved references and the **U** option is on, the unresolved references are printed on the console along with the name of the module in which the linker first encountered the reference, and the user is prompted for more commands to resolve the references. These commands must be terminated by another **END** command.

If there are unresolved references and the **U** option is not on, the link edit will abort, giving the appropriate error message. If there are no unresolved external references, or unresolved references were resolved via additional user commands, pass 2 begins processing.

4.9 ENTRY

The format of the command is:

ENTRY <name>|[(<seg>)]<address>

where:

<name> is the name of a symbol externally defined in one of the input modules previously processed.

<seg> is a segment name and must be enclosed in parentheses.

<address> is the logical address of the entry point. If specified alone, it is recognized as absolute. If specified with segment name, it is relative to the start of that segment. Whether relative or absolute, the corresponding address must be even.

This command indicates the beginning execution address of the S-record module, load module, or relocatable object module being produced. If the <name> parameter is used, the point at which that symbol is defined will then be the entry point of the load module. If a relocatable object module is being produced, this is the only way the entry point may be specified. However, if an S-record or load module is being created, the entry point may also be specified by giving an address.

If an **ENTRY** command is not given, the first entry point specification encountered in the object module input will be used as the beginning execution address of the output module. However, if no starting address is encountered when a load module or S-record module is being created, the beginning address of the first (lowest order) segment will be used as the entry point to the module. Otherwise, no entry point specification will be put in a resulting relocatable object module.

EXAMPLES:

ENTRY START

The symbol START, an **XDEF** defined in one of the relocatable object module inputs, is to be the beginning execution address of the resultant load module.

ENTRY (SEGO)$1214

Relative address $1214 in a segment named SEGO is to be used as the beginning execution address of the load module being created.

ENTRY 16K

The starting execution address of the resultant load module is to be logical address $4000 (16Kb).

4.10 IDENT

This command may be used only if a relocatable object module or S-record format module is being created (the **R** or **Q** option is on). It is used to specify the header identification information to go in that module. For relocatable object modules, this information will be displayed in later linkage edits that use this module and have the **H** option turned on. In the S-record, the identification information is placed in the S0 record.

The format of the command is:

IDENT <mname>,<ver>,<rev>[,<description>]

where:

<mname> is the module name. If a relocatable object module is being created, this is the name by which the module will be referred in later linkage edits.

<ver> and <rev> are the version and revision numbers, respectively, of the module. Both numbers must be integers between 0 and 255, inclusive.

<description> is a description of the module. It may consist of any printable characters (including spaces) and may be up to 80 characters long (relocatable object modules) or 36 characters long (S-record modules).

The module name and version and revision numbers are required, whereas the description is optional.

If an **IDENT** command is not specified, the following defaults will be used:

a. The module name will be the name of the output file being created.
b. The version and revision numbers will both be 1.
c. There will be no description.

EXAMPLES:

IDENT REALSUBS,1,1,REAL NUMBER PACKAGE - RWM

The module name of the relocatable object module being created is REALSUBS. The version and revision numbers will both be one. The description is REAL NUMBER PACKAGE - RWM.

IDENT MODULEA,2,0

The module name will be MODULEA. The version number will be 2 and the revision will be 0. There will be no description.

4.11 INPUT

The format of the command is:

IN[PUT] <fn>[<<mname>[,<mname>]...>][,<fn>[<<mname>[,<mname>]...>]]...

where:

<fn> is a filename.

<mname> is a module name qualifying the <fn> which precedes it. The module name or series of names following each <fn> must be enclosed in angle brackets (< >).

The **INPUT** command extends (or replaces) the input filename capability of the command line. Only filenames can be specified on the command line; here individual module names, as well as filenames, may be specified. (Files are entry names in the operating system directory; modules are file subsets, whose names were specified in assembler **IDNT**, Pascal **PROGRAM** or **SUBPROGRAM**, or linker **IDENT** directives.)

Using **INPUT** commands, in addition, permits library searching (refer to paragraph 4.12) to be interspersed with reading and processing of files.

The following rules apply to **INPUT** commands:

a. The default extension for input filenames is .RO.

b. When a file is not qualified using module names, the entire file is processed.

c. When module names qualify a file, only those modules are processed. The modules will be processed in the order they appear in the object file; this may or may not be the order specified in the **INPUT** command.

d. Data in the output module appears in the same order it appears in the object files requested by the command line and/or **INPUT** commands.

e. If all filenames, or module names, do not fit on a single **INPUT** command line, multiple commands may be used. However, the parameter list after each **INPUT** command must start with a filename.

f. Files in **INPUT** commands are read after any input files on the **LINK** command line.

EXAMPLES:

```
INPUT  VOL1:..MATH.RL,TERMIO,VOL2:..CALC
```

This command causes processing to be performed on the files VOL1:..MATH.RL, TERMIO, and VOL2:..CALC.RO, in that order.

```
IN  FIB,VOL1:..MATH.RL<MULT,DIV>,VOL2:..CALC<MODULE1>
```

All the relocatable object modules from the file FIB.RO, along with the modules MULT and DIV from the file VOL1:..MATH.RL, and the module MODULE1 from the file VOL2:..CALC.RO will be processed.

4.12 LIBRARY

The format of the command is:

LIB[RARY] <lib>[,<lib>]...

where:

<lib> is the name of a library file. If not specified, an extension of .RO is assumed.

The library files specified are searched in the order listed. Only the modules in each file that contain definitions of up-to-now unresolved external references will be processed. Once a definition is found for an unresolved reference, that reference does not play any further part in the searching of libraries. The libraries are searched as soon as the **LIB** input line is ended, before processing more **INPUT** files.

The **LIB** command is normally used when:

a. The **L** option was not used in the command line.

b. Some variation is required in the file/library input specification.

c. Unresolved external references exist at the end of pass 1 and the **U** option is on.

EXAMPLE:

LIB VOL1:..PASCLIB,VOL2:..SYSLIB.SY,MYLIB

This command will cause the library files VOL1:..PASCLIB.RO, VOL2:..SYSLIB.SY, and MYLIB.RO to be searched, in that order, in an attempt to resolve the current unresolved external references.

4.13 LIST

The **LIST** commands allow the user to output various information about the link edit during the interactive session. The command has several forms:

LIST <fn>|#|**#PR**|**#PR**n
LISTM
LISTU
LISTX

where:

<fn> is a disk filename, # specifies the console, and **#PR** or **#PR**n specifies the printer.

The **LIST** command directs all listings produced by the **LISTM**, **LISTU**, and **LISTX** commands to the disk file or device specified by <fn>. If a listing file or device has been specified by a previous **LIST** command, that file is closed and the new file is used until a new file is specified, or the link edit session is ended. If list commands are used without first specifying a listing file, the default output file/device (usually the user's console) will be used.

The **LISTM** command will produce an immediate listing of the current load map (refer to paragraph 5.2, format #6). Listing the load map during pass 1 does not cause any memory allocation to take place and, therefore, no absolute addresses (except for the starting and ending addresses of absolute sections) are printed. Instead, the load map shows what sections have been encountered, what their sizes are, and what symbols are defined in them.

The **LISTU** command produces an immediate listing of all current unresolved external references (refer to paragraph 5.2, format #8).

The **LISTX** command will produce an immediate listing of the current external definition dictionary (refer to paragraph 5.2, format #7).

The use of the list commands does not affect those listings produced by the specification of options in the **LINK** command line. In other words, if any options in the command line indicate that listings are to be produced, they will be produced (at the end of the link edit process), and put in the listing file specified on the command line. The use of list commands is only local to the interactive session.

EXAMPLES:

LIST VOL1:..LISTINGS.SA

This command indicates that all output produced by subsequent list commands is to go in the file VOL1:..LISTINGS.SA.

LIST #PR

This command will cause all output produced by subsequent list commands to go to the line printer.

4.14 MONITOR

The **MONITOR** command specifies the name and session number of a monitor task for the task being created. Its form is:

MON[ITOR] <name> [,<session number>]

where:

<name> is a one- to four-character taskname that conforms to the same rules as those for segment names.

<session number> is optional; if specified, it may take one of two forms:

a. A number in the range $0 to $FFFFFFFF, inclusive.

b. A single quote ('), followed by a number in the range $0 to $FFFF, inclusive.

If the first form is used, the session number will be put in the load module in pure binary form. This usually indicates a special system session number. However, if the second form is specified, the session number in the load module will be the hex value of the input session number encoded in ASCII. For example, '$45AF is put into the load module as $34354146. This form usually indicates a regular user session number.

If a session number is not specified, the session number defaults to binary zero.

If the **MONITOR** command is not specified, both the monitor name and session number in the load module default to binary zeros.

EXAMPLES:

```
MONITOR MON1,'$ABCD
MON HANK
MON _&$.,%1011100011011011010
```

This command cannot be used when a relocatable object module or an S-record module is being created.

4.15 OPTIONS

The **OPTIONS** command specifies the task directive options. Its form is:

OPT[IONS] <options>

where:

<options> is a list of zero or more one-character options, which may or may not be separated by spaces.

The legal options and their meanings are:

M A monitor is specified.

P Propagate the monitor, use the monitor of the loading task.

If an **OPTIONS** command is specified with no options, or no **OPTIONS** command is specified, then no directive options will be put in the load module.

EXAMPLES:

```
OPTIONS MP
OPT
OPT P M
OPT P
```

This command may not be used when a relocatable object module or an S-record module is being created.

4.16 PAGESIZE

The **PAGESIZE** <number> interactive user command enables modification of the page size of the load module from the default value of 256 to any even value in the range 256 <= <number> <= 32766. All segment starting addresses in the used segment allocation descriptors are rounded up to the next multiple of <number>. (Refer to Appendix G for MVME12x-specific information.)

4.17 PRIORITIES

The **PRIORITIES** command specifies the initial and limit priorities of the task being created. Its syntax is:

PRIO[RITIES] <initial priority>,<limit priority>

where:

<initial priority> specifies the initial priority of the task.

<limit priority> specifies its limit priority.

Both priorities must be specified and both must be numbers between 0 and 255, inclusive.

If the **PRIORITIES** command is not given, both priorities default to zero.

EXAMPLES:

```
PRIORITIES 0,255
PRIO $10,$20
```

This command cannot be used when a relocatable object module or an S-record module is being created.

4.18 QUIT

QUIT signals the end of user commands. It is used exactly like the **END** command and produces the same results. (Refer to paragraph 4.8.)

4.19 SEGMENT

The format of the command is:

SEG[MENT] <seg>[(<attr>)]:<sec#>[,<sec#>]...[<start>[,<end>]]

where:

<seg> is a segment name of up to four characters, the first of which must be alphabetic; the remainder may be alphanumeric.

<attr> if present, specifies the attributes of the segment, and may be any combination of the letters **R**, **L**, and **G**. If a letter is not specified, the attribute is turned off. The letters mean:

R Segment is read only.
L Segment is locally shareable.
G Segment is globally shareable.

<sec#> is either a single section number or a range of section numbers. A range of sections is specified by giving a section number, followed by a dash and a second section number greater than the first section number. All section numbers are between 0 and 15, inclusive. Section numbers are loaded into the segment in the order specified in the **SEGMENT** command; however, this may be overridden by subsequent **START** commands.

<start> if specified, starts the segment at that logical address; if starting addresses are not defined, segments are located in the order they are defined by **SEGMENT** commands.

<end> is the ending address. If <end> is not given, the segment will be as long as is necessary to hold all sections that are to go in the segment.

If a load module or S-record module is being produced, the **SEGMENT** command is used to define an MMU. The MMU dictates that segment sizes be a multiple of 256 bytes. To ensure this, the low order byte must be $00 when specifying a starting address of a segment. In conjunction with this, the low order byte of ending addresses must be $FF.

If **SEGMENT** commands are specified, they must appear before any **START**, **INPUT**, or **LIB** commands. Therefore, when using **SEGMENT** commands, input files may not be specified in the **LINK** command line.

When **SEGMENT** commands are specified, only those section numbers indicated in the **SEGMENT** commands are loaded. If, during the processing of a relocatable object module, a section is encountered with a number not specified in a **SEGMENT** command, that section will not be processed, and a warning message will be generated. Any symbols that are defined in unassigned sections will not be loaded into the symbol table, and warning messages will be generated as they are encountered.

SEGMENT

If no **SEGMENT** commands are specified before the first **INPUT**, **START**, or **LIB** command, three segments (named SEG0, SEG1, and SEG2) will automatically be set up to contain all sections. The sections are assigned as follows: SEG0 contains sections 0 through 7, SEG1 contains sections 8 through 14, and SEG2 contains section 15. In addition, SEG1 is assigned the read-only attribute. The other segments will have no attributes assigned to them. Sections are loaded into their respective segments in order of increasing section number. The segments are assigned memory in the order in which they are encountered, with each segment as long as necessary to hold the sections assigned to it. (Refer to Appendix G for MVME12x-specific information.)

NOTE

If the **LINK** command line is used alone, short sections should be assigned to the lowest order section numbers being used; this will ensure that the short sections are allocated at the beginning of low memory.

EXAMPLES:

```
SEGMENT SEG0:1,15,6-9 $30100, $312FF
```

This command assigns sections 1, 15, 6, 7, 8, and 9, in that order, to a segment named SEG0, which starts at logical address $30100 and ends at logical address $312FF.

```
SEGMENT SEG2(R):12,10,2 45K
```

This command assigns sections 12, 10, and 2 to a segment named SEG2, which is designated as read only. Finally, it starts at logical address $B400 (45K), and will be as long as necessary to contain all the data assigned to it.

```
SEGMENT SEG3(LG):14
```

This command assigns one section, section 14, to a segment named SEG3, which will be locally and globally shareable. This segment will start wherever there is room, and will be just as long as necessary.

When a Pascal program which is to be run on a KDM, a VERSAmodule 01 with VERSAbug, or a simulator is being linked, the **SEGMENT** command must be used to prevent the program from being loaded into low memory space that is reserved for the system. The following commands should be included:

```
SEG   SEG0:1-7 $0000,$0FFF
SEG   SEG1(R):8-14
SEG   SEG2:0,15
```

Further explanation of these requirements are found in the M68000 Family Resident Pascal User's Manual.

4.20 START

If a load module or S-record module is being created, this command is used to define the starting address at which a particular section or sections will be stored. The format of the command is:

START <sec#>[,<sec#>]... <address>

where:

<sec#> is either a single section number or a range of section numbers. A range of sections is specified by giving a section number, followed by a dash and a second section number greater than the first section number. All section numbers are between 0 and 15, inclusive. If more than one section number is specified, the first section will start at the given address and the remaining sections will immediately follow in the order specified. The order of sections within a segment may be changed in this way.

<address> specifies the address at which to start putting the given sections. It is either an absolute logical address or a relative address, depending on the following conditions:

a. If no **SEGMENT** commands were issued or a starting address was specified in the **SEGMENT** command for the segment containing the section(s) in the **START** command, <address> will be interpreted as an absolute logical address.

b. Otherwise, if the starting address was not given in the **SEGMENT** command for the segment containing the sections in the **START** command, <address> will be interpreted as being relative to the start of the segment.

It is an error if all the sections specified in a particular **START** command do not belong to the same segment.

EXAMPLE:

START 10-15,1 $1422E

This command indicates that sections 10, 11, 12, 13, 14, 15, and 1 are to be stored in that order, starting at address $1422E. This address will be interpreted as absolute or relative to the beginning of the segment that contains sections 10-15 and 1, depending on the criteria mentioned above.

4.21 TASK

The **TASK** command sets up the name and session number of an applications program which will be executed as a "task". (Refer to the M68000 Family Real-Time Multitasking Software User's Manual.) The command syntax is:

TASK <name>[,<session number>]

where:

<name> is a one- to four-character taskname that conforms to the same rules as those for segment names.

<session number> is optional; if specified, it may take one of two forms:

a. A number in the range $0 to $FFFFFFFF, inclusive.

b. A single quote ('), followed by a number in the range $0 to $FFFF, inclusive.

If the first form is used, the session number will be put in the load module in pure binary form. This usually indicates a special system session number. However, if the second form is specified, the session number in the load module will be the hex value of the input session number encoded in ASCII. For example, '$45AF is put into the load module as $34354146. This form usually indicates a regular user session number.

If a session number is not specified, the session number defaults to binary zero.

If the **TASK** command itself is not specified, the taskname in the load module defaults to the first four characters of the name of the load module file being created, and the session number is binary zeros.

EXAMPLES:

```
TASK &TK1,$12C4BF
TASK DIR
TASK LINK,'152
```

This command may not be used when a relocatable object module or S-record module is being created.

4.21 XDEF

The format of the command is:

XDEF <symbol>[,<symbol>]...

where:

<symbol> is an externally defined symbol in a relocatable object module that has already been processed.

If a relocatable object module is being produced (via the **R** option), this command may be used to indicate which symbols are to be externally defined in the new object module. In other words, those symbols that may be referenced by other modules.

When the linker is used to create a relocatable object module, all the externally defined symbols encountered in the input modules will be carried along and defined as **XDEF**s in the new module. However, if an **XDEF** command is given, only those symbols in the **XDEF** command will be externally defined in the new object module. More than one **XDEF** command may be given, in which case all the symbols in all the **XDEF** commands, and only those symbols, will be **XDEF**s in the new object module.

EXAMPLE:

XDEF ENTRY1,.XDIV,MAIN

The command indicates that the symbols ENTRY1, .XDIV, and MAIN are to be externally defined in the relocatable object module being created.

4

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 5

LISTING FORMATS

5.1 GENERAL

At the end of pass 1 when unresolved references still exist, at a fatal error, at the end of pass 2, or immediately in response to user listing commands, the linker will print listings (refer to Table 5-1). Some of the listings at the end of pass 2, and at fatal errors, occur only as a result of options specified in the **LINK** command line. The listings are directed to the listing output file/device (refer to paragraphs 3.3 and 4.13). Paragraph 5.2 contains syntactical formats for the twelve listings. Paragraph 5.3 contains example printouts.

TABLE 5-1. Listing Occurrences

TIME OF OCCURRENCE	PRINTOUT FORMATS #1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
IMMEDIATELY							LISTM	LISTX	LISTU			
END OF PASS 1								(+)				
END OF PASS 2	+	I	+	I	H	M	X	+	+	O,Q	+	+
FATAL ERROR	+	I	+	I				+	+		+	+

NOTE: + means always print out.

(+) means print out at end of pass 1 when unresolved references still exist.

Single letters are command line options; 5-letter words are user commands.

5.2 SYNTACTICAL FORMATS

FORMAT #1

```
M68000 Linkage Editor Version x.xx
Copyrighted 1985 by Motorola, Inc.
```

FORMAT #2

Command Line:

<command line used to invoke the linkage editor>

FORMAT #3

Options in Effect: <list of options>

FORMAT #4

User Commands: [None]

<user command>
<user command>
.
.
.
<user command>

5

FORMAT #5

Object Module Header Information:

Module	Ver	Rev	Language	Date	Time	Creation Filename
<mname>	<v>	<r>	<lang>	<date>	<time>	<filename>
	<description>					
<mname>	<v>	<r>	<lang>	<date>	<time>	<filename>
	<description>					
					.	
					.	
					.	
<mname>	<v>	<r>	<lang>	<date>	<time>	<filename>
	<description>					

FORMAT #6

```
Load Map:

Segment <sg name>[(<attr>)]: <start adr> <end adr> <list of sections in sg>

Module    S   T  Start     End        Externally Defined Symbols

<mname>  <s> <t> <start>   <end>      <sname> <address>   <sname> <address>
                                      <sname> <address>   <sname> <address>
                                                    .
                                                    .
                                                    .
                                      <sname> <address>   <sname> <address>
<mname>  <s> <t> <start>   <end>      <sname> <address>   <sname> <address>
                                .
                                .
                                .
<mname>  <s> <t> <start>   <end>      <sname> <address>   <sname> <address>

Segment <sg name>[(<attr>)]: <start adr> <end adr> <list of sections in sg>

Module    S   T  Start     End        Externally Defined Symbols

<mname>  <s> <t> <start>   <end>      <sname> <address>   <sname> <address>

                                .
                                .
                                .
Segment <sg name>[(<attr>)]: <start adr> <end adr> <list of sections in sg>

Module    S   T  Start     End        Externally Defined Symbols

<mname>  <s> <t> <start>   <end>      <sname> <address>   <sname> <address>
```

FORMAT #7

```
Table of Externally Defined Symbols:

Name      Address    Module     Displ      Sect Seg  Library       Input

<sname>  <abs adr>  <mname>    <rel adr>   <s> <sg> <lib name>    <inp name>
<sname>  <abs adr>  <mname>    <rel adr>   <s> <sg> <lib name>    <inp name>
                                  .
                                  .
                                  .
<sname>  <abs adr>  <mname>     <rel adr>  <s> <sg> <lib name>    <inp name>
```

FORMAT #8

```
Unresolved External References:  [None]

<sname>     <sname>     <sname>     <sname>     <sname>     <sname>
<sname>     <sname>     <sname>     <sname>     <sname>     <sname>
                                       .
                                       .
                                       .
<sname>     <sname>     <sname>     <sname>     <sname>     <sname>
```

FORMAT #9

```
Multiply Defined Symbols:  [None]

<sname>     <sname>     <sname>     <sname>     <sname>     <sname>
<sname>     <sname>     <sname>     <sname>     <sname>     <sname>
                                       .
                                       .
                                       .
<sname>     <sname>     <sname>     <sname>     <sname>     <sname>
```

5

FORMAT #10

```
Length (in bytes):

       Segment     Hex          Decimal

       <sg name>  <hex length>  <decimal length>
       <sg name>  <hex length>  <decimal length>
                             .
                             .
                             .
       <sg name>  <hex length>  <decimal length>
Total Length:     <hex length>  <decimal length>
```

FORMAT #11

```
<error count>     Error(s)
                  <error msg>
                  <error msg>
                       .
                       .
                       .
                  <error msg>

<warning count>   Warning(s)
                  <warning msg>
                  <warning msg>
                       .
                       .
                       .
                  <warning msg>
```

FORMAT #12

```
Load module: <output filename> has been created.|
Load module: <output filename> has been replaced.|
Load module has not been created due to fatal error.|
Relocatable object module: <output filename> has been created.|
Relocatable object module: <output filename> has been replaced.|
Relocatable object module has not been created due to fatal error.|
S-record module: <output filename> has been created.|
S-record module: <output filename> has been replaced.|
S-record module has not been created due to fatal error.|
```

5.2.1 Meaning of Symbols in Output Format

Symbol	Meaning
<mname>	Module name or name of common section
<v>	Version number
<r>	Revision number
<lang>	Source language (for example, Assembly, FORTRAN, or Pascal)
<filename>	Name of source file used to create object module
<date>	Date of creation of module
<time>	Time of creation of module
<description>	Description of module
<attr>	Attributes of segment
<start adr>	Starting address of segment
<end adr>	Ending address of segment
<s>	Section number
<t>	Type of section: <blank> - Standard load section S - Short load section A - Absolute section C - Common section
<start>	Starting address of section
<end>	Ending address of section
<sname>	Symbol name
<address>	Absolute address of symbol
<abs adr>	Absolute address of symbol
<rel adr>	Relative address of symbol within section
<sg>	Segment number
<lib name>	Name of library file in which symbol is defined
<input name>	Name of regular input file in which symbol is defined

The rest of the symbols meanings are self-explanatory.

5.3 EXAMPLE PRINTOUTS

5.3.1 Example Output #1

```
M68000 Linkage Editor Version x.xx
Copyrighted 1985 by Motorola, Inc.

Command Line:

LINK ;MX1HO

Options in effect:  A,-B,-D,-H,1,-L,M,O,P,-Q,-R,-S,-U,-W,X

User Commands:

SEGMENT SEG1(RL):0-2 0,$FBFF
SEGMENT SEG0:5,3
SEGMENT SEG2:4 64K
START 2 $A000
START 3 $20000
INPUT MAIN,SUBRTS
INPUT MATH<ADD,SUB,MULT,DIV>
LIB INOUT.LB
ENTRY START
END


Object Module Header Information:

Module      Ver Rev Language Date     Time     Creation Filename

MAIN          1   0 Pascal   10/24/80 01:12:56 VOL1:14.CATALOG1.FIB.SA
            FIBONACC1 NUMBER GENERATOR - MAIN PROGRAM - JRK

SPROGS        1   1 Fortran  10/25/80 11:01:32 VOL1:14.CATALOG2.SUBPROGS.SA
            FIBONACC1 NUMBER GENERATOR - SUBPROGRAMS - JRK

ADD           1   1 Assembly 09/27/80 09:12:34 VOL2:25.CATA.FPADD.SA
            FLOATING POINT ADD - DRM

SUB           2   0 Assembly 09/27/80 09:21:18 VOL2:25.CATA.FPSUB.SA
            FLOATING POINT SUBTRACT - DRM

MULT          2   2 Assembly 09/27/80 09:15:16 VOL3:56.MYCAT.FPMULT.SA
            FLOATING POINT MULTIPLY - GDK

DIV           1   0 Assembly 09/27/80 09:31:02 VOL3:56.MYCAT.FPDIV.SA
            FLOATING POINT DIVIDE - GDK

TERMIO        1   2 Assembly 07/01/80 14:01:52 VOL4:119.IOCAT.TERMINAL.SA
            TERMINAL I/O - DWH

DISKIO        1   2 Assembly 07/04/80 16:59:59 VOL1:7.IOCAT.DISK.SA
            DISK I/O - CRF
```

Load Map:

Segment SEG0: 00025500 00045AFF 5,3

Module	S	T	Start	End	Externally Defined Symbols	
ADD	5		00025500	00025711	.XADD	00025500
SUB	5		00025712	00025B67	.XSUB	00025712
MULT	5		00025B68	00025F73	.XMULT	00025B68
DIV	5		00025F74	0002646F	.XDIV	00025F74
MAIN	3		00045500	0004551F		
SPROGS	3		00045520	0004555B		
COMM1	3	C	0004555C	0004575B		
COMM2	3	C	0004575C	00045A5B		

Segment SEG1(R,L): 00000000 000FBFF 0,1,2

Module	S	T	Start	End	Externally Defined Symbols			
MAIN	0	S	00000000	000000FB				
SPROGS	0	S	000000FC	0000011B	AREG	000000FC	BREG	00000100
					CREG	00000104	XREG	00000108
					YREG	0000010C	ZREG	00000110
MAIN	1		0000011C	000025CB				
SPROGS	1		000025CC	00003623				
.BLANK	1	C	00003624	00003723				
MAIN	2		0000A000	0000B0FF				
SPROGS	2		0000B100	0000D0FF	SUB1	0000B100	SUB2	0000BC92
					SUB3	0000C34E	SUB4	0000D000

Segment SEG2: 00010000 000254FF 4

Module	S	T	Start	End	Externally Defined Symbols			
MAIN		A	00010000	0001356B				
MAIN	4		0001356C	000141FF	START	0001356C		
SPROGS	4		00014200	00015433				
TERMIO	4		00015434	00019131	TIN	00015434	TOUT	0001732C
DISKIO	4		00019132	00025433	OPEN	00019214	CLOSE	00019466
					READ	0001CDFE	WRITE	0002433E

Table of Externally Defined Symbols:

Name	Address	Module	Displ	Sect	Seg	Library	Input
.XADD	00025500	ADD	00000000	5	SEG0		MATH.RO
.XDIV	00025F74	DIV	00000000	5	SEG0		MATH.RO
.XMULT	00025B68	MULT	00000000	5	SEG0		MATH.RO
.XSUB	00025712	SUB	00000000	5	SEG0		MATH.RO
AREG	000000FC	SPROGS	00000000	0	SEG1		SUBRTS.RO
BREG	00000100	SPROGS	00000004	0	SEG1		SUBRTS.RO
CLOSE	00019466	DISKIO	00000334	4	SEG2	INOUT.LB	
CREG	00000104	SPROGS	00000008	0	SEG1		SUBRTS.RO
OPEN	00019214	DISKIO	000000E2	4	SEG2	INOUT.LB	
READ	0001CDFE	DISKIO	00003CCC	4	SEG2	INOUT.LB	
START	0001356C	MAIN	00000000	4	SEG2		MAIN.RO
SUB1	0000B100	SPROGS	00000000	2	SEG1		SUBRTS.RO
SUB2	0000BC92	SPROGS	00000B92	2	SEG1		SUBRTS.RO
SUB3	0000C34E	SPROGS	0000124E	2	SEG1		SUBRTS.RO
SUB4	0000D000	SPROGS	00001F00	2	SEG1		SUBRTS.RO
TIN	00015434	TERMIO	00000000	4	SEG2	INOUT.LB	
TOUT	0001732C	TERMIO	00001EF8	4	SEG2	INOUT.LB	
WRITE	0002433E	DISKIO	0000B20C	4	SEG2	INOUT.LB	
XREG	00000108	SPROGS	0000000C	0	SEG1		SUBRTS.RO
YREG	0000010C	SPROGS	00000010	0	SEG1		SUBRTS.RO
ZREG	00000110	SPROGS	00000014	0	SEG1		SUBRTS.RO

Unresolved External References: None

Multiply Defined Symbols:

CLOSE ZREG START

Lengths:

Segment	Hex	Decimal
SEG0	00020600	132608
SEG1	0000FC00	64512
SEG2	00015500	87296
Total Length	00045700	284416

No Errors
3 Warnings

** Warning 701 - Multiply Defined Symbol: CLOSE
** Warning 701 - Multiply Defined Symbol: ZREG
** Warning 701 - Multiply Defined Symbol: START

Load module :MAIN.LO has been created.

5.3.2 Example Output #2

```
M68000 Linkage Editor Version x.xx
Copyrighted 1985 by Motorola, Inc.


Command Line:

LINK MAIN/SUBRTS,,#PR;-PO


Options in Effect:  -A,-B,-D,-H,-I,-L,-M,O,-P,-Q,-R,-S,-U,-W,-X


Unresolved External References:

.XRESET      .XREWRITE    .XREAD       .XWRITE      .XSQRT       .XEOF
.XEOLN       .XEXIT


Multiply Defined Symbols:  None


     1  Error

        ** ERROR 600 - Unresolved References

    No  Warnings


Load module has not been created due to fatal error.
```

5.3.3 Example Output #3

```
M68000 Linkage Editor Version x.xx
Copyrighted 1985 by Motorola, Inc.


Command Line:

LINK ,OUTPUT,#PR;IXHMR


Options in Effect:  A,-B,-D,H,I,-L,M,-O,-P,-Q,R,-S,-U,-W,X
```

```
User Commands:

IN PASS1TF2
LISTX
IN PASS1TF1
LISTU
LISTX
LISTM
LISTX
XDEF SYMBOL111,SYMBOL202,SYMBOL1A1,SYMBOL343
IDENT MODULE12,1,1,THIS IS A COMBINATION OF MODULE 1 AND MODULE 2
END


Object Module Header Information:

Module     Ver Rev Language Date     Time     Creation Filename

MODULE2      1  10 BASIC
           THIS IS MODULE NUMBER 2

MODULE1      1   1 ASSEMBLY
           THIS IS MODULE NUMBER 1


Load Map:


Relocatable Sections:

Module      S   T  Start     End       Externally Defined Symbols

MODULE2         0        00000000  00000029  SYMBOL201  00000000  SYMBOL202
0000001A
MODULE1         0        0000002A  00000129  SYMBOL101  0000002A  SYMBOL102
0000005A
COMMON1     0   C  00000000  000000FF
MODULE2     1      00000000  00000081  SYMBOL211  00000000
MODULE1     1      00000082  0000014D  SYMBOL111  00000082
MODULE2     2   S  00000000  0000001B
MODULE1         2   S   0000001C  0000004F  SYMBOL121  0000001C  SYMBOL122
0000002E
                                          SYMBOL123  00000040
COMMON2     2   C  00000000  0000002F
MODULE2     3      00000000  0000004F
MODULE1     5      00000000  00000111
MODULE1     6   S  00000000  00000123
MODULE2     9   S  00000000  000000FD
MODULE2    13      00000000  00000203
MODULE1    13      00000204  0000023B
COMMON4    13   C  00000000  0000000F
```

```
Absolute Sections:

Module      S   T  Start     End       Externally Defined Symbols

MODULE1         A  00000300  000003FF  SYMBOL1A1  00000342
MODULE1         A  00001000  0000I019
MODULE1         A  00002012  00003123  SYMBOL1A2  00003000
MODULE2         A  00009FBC  0000ABDF  SYMBOL2A1  0000AAAA


Table of Externally Defined Symbols:

Name        Address   Module      Displ       Sect Seg  Library     Input

SYMBOL111   00000082  MODULE1     00000000     1                    PASS1TF1.RO
SYMBOLIA1   00000342  MODULE1     00000042                          PASS1TF1.RO
SYMBOL202   0000001A  MODULE2     0000001A     0                    PASS1TF2.RO


Unresolved References

SYMBOL301    SYMBOL3151   SYMBOL3152   SYMBOL381    SYMBOL382    SYMBOL4151

Multiply Defined Symbols: None


   No Errors
    1 Warning

      ** WARNING 700 - UNDEFINED SYMBOL: SYMBOL343

Relocatable object module :OUTPUT.RO has been created.
```

5.3.4 Example Output #4

```
M68000 Linkage Editor Version x.xx
Copyrighted 1985 by Motorola, Inc.


Command Line:

LINK ,OUTFIL,OUTPUT;IAMQ


Options in Effect:  A,-B,-D,-H,I,-L,M,-O,P,Q,-R,-S,-U,-W,-X

User Commands:

IN OBJECTF,OBJECT
END
```

Load Map:

Segment SEG1<R>: 00000000 000016FF 8,9,10,11,12,13,14

MODULE	S	T	Start	End	Externally Defined Symbols			
INIT	8		00000000	00000379	.PLJSR	00000372		
TRAPS	8		0000037A	00000459	.PADDRER	00000442	.PVBUSER	0000042A
					.PVCHKI	00000412	.PVTRAPD	00000412
					.PVTRAPE	0000037A	.PVTRAPV	000003FA
					.PVZDIV	000003E2		
OPTION	8		0000045A	0000053F	.POPTION	0000045A		
CLSCOD	8		00000540	000005BF	.PCLSCOD	00000540		
ALSTS	8		000005C0	000005E5	.PALSTS	000005C0		
CLO	8		000005E6	00000601	.PCLO	000005E6		
IFD	8		00000602	000007F3	.PIFD	00000602		
RST	8		000007F4	000008A5	.PRST	000007F4		
RWT	8		000008A6	00000925	.PRWT	000008A6		
ACCPER	8		00000926	00000947	.PACCPER	00000926		
CALCLU	8		00000948	00000979	.PCALCLU	00000948		
EDTFIL	8		0000097A	00000CFB	.PEDTFIL	0000097A		
PRGBUF	8		00000CFC	00000D15	.PPRGBUF	00000CFC		
STDFLT	8		00000D16	00000D73	.PSTDFLT	00000D16		
DFLT	8		00000D74	00000DCF	.PDFLT	00000D74		
WLN	8		00000DD0	00000DDF	.PWLN	00000DD0		
WRI	8		00000DE0	00000E6D	.PWRI	00000DE2	.PWRH	00000DE0
					.PWRJ	00000DE4		
WRSWRV	8		00000E6E	00000EB7	.PWRS	00000E78	.PWRV	00000E6E
WRTBUF	8		00000EB8	00000F1B	.PWRTBUF	00000EB8		
LBLKS	8		00000F1C	00000F2D	.PLBLKS	00000F1C		
IWPTR	8		00000F2E	00000F41	.PIWPTR	00000F2E		
RLN	8		00000F42	00000F7D	.PRLN	00000F42		
RDI	8		00000F7E	00000F87	.PRDI	00000F7E		
RDJ	8		00000F88	00000F91	.PRDJ	00000F88		
RDINT	8		00000F92	00000FF3	.PRDINT	00000F92		
SBLKS	8		00000FF4	00001005	.PSBLKS	00000FF4		
GETCH	8		00001006	00001023	.PGETCH	00001006		
IRPTR	8		00001024	00001041	.PIRPTR	00001024		
GETINT	8		00001042	00001159	.PGETINT	00001042		
MAKINT	8		0000115A	000011B7	.PMAKINT	0000115A		
RDBUF	8		000011B8	000012A5	.PRDBUF	000011B8		
ASGNF	8		000012A6	000012C9	.PASGNF	000012A6		
BUFSZ	8		000012CA	000012D9	.PBUFSZ	000012CA		
CLOSE	8		000012DA	0000130B	.PCLOSE	000012DA	.PCLOSPL	000012FA
CFLDAD	8		0000130C	00001333	.PCFLDAD	0000130C		
FLSCN	8		00001334	0000135D	.PFLSCN	00001334		
LDC	8		0000135E	00001387	.PLDCS	0000135E	.PLDCV	00001362
FINIT	8	C	00001388	00001389				
FORCE	9		0000138A	000013BD	FORCE	0000138A		
SORT	9		000013BE	000016B9	.PMAIN	000013BE		

Segment SEG2: 00001700 000078FF 15

Module	S	T	Start	End	Externally Defined Symbols	
SORT	15		00001700	000078FF	.PZMAIN	000078FE

```
Unresolved References:  None

Multiply Defined Symbols:  None

Lengths (in bytes):

     Segment       Hex         Decimal

       SEG1      00001700          5888
       SEG2      00006200         25088
Total Length     00007900         30976

    No Errors
    No Warnings

S-record module :OUTFIL.MX has been created.
```

5

THIS PAGE INTENTIONALLY LEFT BLANK.

A

APPENDIX A

RELOCATABLE OBJECT MODULE FILE FORMAT

Relocatable Object Module Storage Format

The VERSAdos operating system stores relocatable object modules in sequential files with fixed length records of 256 bytes. (Refer to the VERSAdos Data Management Services and Program Loader User's Manual for information on the file formats supported by the operating system.)

Within each 256-byte record a variable number of variable length relocatable object records are stored. Each one of these records consists of a 1-byte byte count followed by the actual data of the relocatable record. The byte count indicates the number of data bytes that follow in the record. The byte count may contain any value between 0 and 255, inclusive. A byte count of zero indicates a relocatable object record with no data bytes (a record of this type is ignored by the linkage editor). Thus, the length of one relocatable object record is limited to a total of 256 bytes; one byte for the byte count and a maximum of 255 data bytes.

However, this does not mean that only one variable length relocatable object record can be stored in one fixed length record. Each 256-byte fixed length record is totally filled before continuing to the next 256-byte record. Thus, it is possible for a variable length record to be divided between two fixed length records. For example, suppose the first three relocatable object records of a relocatable object module contained 50, 150, and 200 bytes of data, respectively. The first two records, along with their byte counts, would be stored within the first 202 bytes of the first 256-byte fixed-length record. This would leave 54 bytes remaining in that record. These 54 bytes would be filled with the byte count of the third relocatable object record, followed by the first 53 data bytes of that record. The remaining 147 data bytes of the third relocatable object record would then be stored at the beginning of the second 256-byte fixed length record.

Any space not used in the last 256-byte fixed length record of a relocatable object module file must be filled with binary zeros. This fills out the rest of the file with relocatable object records that have zero bytes of data (which are ignored by the linkage editor).

Relocatable Object Record Format

There are four basic types of relocatable object records. The record type is indicated by the first byte of data (the byte immediately after the byte count), in the record. This byte is the ASCII code of one of the digits between "1" and "4", inclusive. The byte values and the types of records are:

Value of First Data Byte	Record Type
1 ($31)	Identification Record
2 ($32)	External Symbol Definition Record
3 ($33)	Object Text Record
4 ($34)	End Record

A

The formats of these four record types is discussed in detail in the following paragraphs.

Identification Record (Type 1)

Each relocatable object module must contain an identification record as the first record in the module. It is this record that indicates the beginning of a relocatable object module. The identification record contains general information about the relocatable object module, such as its name, version and revision, what language processor was used to create the module, what source file was used to create the module, the time and date the module was created, and a description of the module. The format of an identification record is:

Bytes	1	1	10	1	1	1	4	2	8	8	2	3	3	0-211
Field	Size	1	Mname	V	R	L	Vol	User	Cat	Fname	Ext	Time	Date	Descr

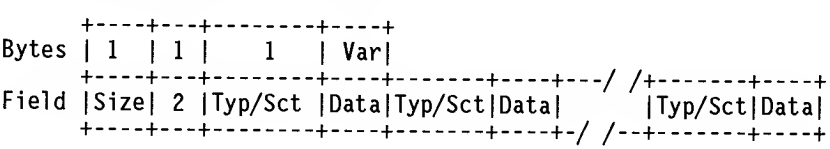
Field	Size (Bytes)	Contents
1	1	Record Type (ASCII $31).
Mname	10	Module name (ASCII).
V	1	Module version number (0-255).
R	1	Module revision number (0-255).
L	1	Language processor type (ASCII): A ($41) - Assembler B ($42) - BASIC C ($43) - COBOL F ($46) - FORTRAN P ($50) - Pascal
Vol	4	Source file volume name (ASCII).
User	2	Source file user number (0-9999).
Cat	8	Source file catalog name (ASCII).
Fname	8	Source file filename (ASCII).
Ext	2	Source file extension (ASCII).
Time	3	Module creation time (hhmmss). (NOTE)
Date	3	Module creation date (mmddyy). (NOTE)

Descr	varies	Module description (ASCII). Occupies remainder of identification record as indicated by record length. May be 0-211 bytes (characters) long.

(NOTE) Time and date are stored in a BCD format with two decimal digits per byte. For example, if the time of creation was 9:27:56, it would be stored in the identification record as $092756.

External Symbol Definition Record (Type 2)

Each external symbol definition record contains a variable number of External Symbol Definitions (ESDs) and defines a relocatable section, a common section, an absolute section, an externally defined symbol, an externally referenced symbol, or a command line address. A 1-byte value at the beginning of the ESD indicates the type of ESD within an external symbol definition record. The high order nibble of the byte indicates the ESD type while the low order nibble of the byte indicates what section the ESD refers to. The format of an external symbol definition record is:



Field	Size (Bytes)	Constants
2	1	Record type (ASCII $32)
Typ/Sct	1	Typ (high nibble) = Type of ESD

0 - Absolute section

1 - Common section (in section Sct)

2 - Standard relocatable section (section number Sct)

3 - Short address relocatable section (section number Sct)

4 - External symbol definition (in relocatable section Sct)

5 - External symbol definition (in an absolute section)

6 - External symbol reference (to section Sct)

7 - External symbol reference (to any section)

A

8 - Command line address (in section Sct)

9 - Command line address (in an absolute section)

A - Command line address (in a common section in section Sct)

Sct (low nibble) = Relocatable section referring to (0-15)

Data	Varies	Depends on Typ (see below)

Several ESD entries may be included in one ESD record. The following descriptions outline the contents of the Data field from the general ESD record format:

Absolute section

Bytes	1	4	4
	0/0	Size	Start

Common section

Bytes	1	10	4
	1/Sct	Common	Size

Standard relocatable section

Bytes	1	4
	2/Sct	Size

Short address relocatable section

Bytes	1	4
	3/Sct	Size

XDEF (in section Sct)

Bytes	1	10	4
	4/Sct	XDEF	Address

XDEF (in an absolute section)

Bytes	1	10	4
	5/0	XDEF	Address

A

XREF (to section Sct)

Bytes	1	10
	6/Sct	XREF

XREF (to any section)

Bytes	1	10
	7/0	XREF

Command line address (in section Sct)

Bytes	1	4	1
	8/Sct	CL Adr	CL Lngth

Command line address (in an absolute section)

Bytes	1	4	1
	9/0	CL Adr	CL Lngth

Command line address (in a common section)

Bytes	1	10	4	1
	A/Sct	CL Com	CL Adr	CL Lngth

Field	Size (Bytes)	Contents
Size	4	Length of section (in bytes).
Start	4	Starting address of absolute section.
Common	10	Name of common section (ASCII).
XDEF	10	Name of **XDEF** symbol (ASCII).
Address	4	Address of symbol within its section (in 5/0 this is an absolute address).
XREF	10	Name of **XREF** common symbol (ASCII).
Cl Adr	4	Address of command line within its section (in 9/0 this is an absolute address).
CL Lngth	1	Maximum length of command line -1. (0-255 represents 1-256.)
CL Com	10	Name of common section that contains the command line (ASCII).

A

ESDs have a restriction that all ESDs defining externally defined and externally referenced symbols (ESD types 4-7) must appear before any other types of ESDs in the module.

Each section (relocatable, common, and absolute) and each external reference in a relocatable object module is assigned an index so that they may be easily referenced later in the relocatable object module. This index is an External Symbol Definition Index (ESDID). Since a module may contain only one of each type of relocatable section (sections 0-15), the ESDID for a relocatable section is simply the section number plus 1. Thus, the index for section 12 is 13. However, a relocatable object module may contain multiple common sections, absolute sections, and external references. Therefore, indices for these types of ESDs are assigned in increasing numerical order, starting at 17, in the order the ESDs are encountered in the module. Thus, the index of the first ESD of the type 0, 1, 6, or 7 is 17; the index of the second ESD of those types is 18; and so on. A Pascal-like algorithm for assigning ESDIDs when reading a relocatable object module is:

```
i  := 17;
WHILE reading ESDs DO
     BEGIN
          read an ESD of Typ/Sct;
          CASE Typ OF
             0,1,6,7:      BEGIN
                             ESDID  := i;
                             i      := i + 1
                           END;
             2,3:          ESDID   := Sct + 1;
             4,5,8,9,10:   {no ESDID assigned};
          END; {CASE}
          process the ESD
     END  {WHILE}
```

where:

ESDID is the ESD index for the ESD that is currently being processed.

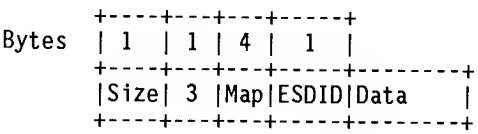
NOTE: ESDs that describe externally defined symbols and command line addresses are not assigned indices. This is because these types of ESDs do not need to be referred to later in the relocatable object module.

New ESDIDs are assigned for each relocatable object module processed. Thus, the ESDIDs in one module have no relation to the ESDIDs in another module. Each module is limited to a total of 255 ESDIDs (numbered 1 through 255). Since ESDIDs 1 through 16 always refer to the relocatable sections 0 through 15, a relocatable object module may contain at most a total of 239 absolute sections, common sections, and external references.

A

Object Text Record (Type 3)

Object text records define the actual code and data to be put in the resulting load module (or relocatable object module). Each object text record contains absolute code along with relocation data for computing relocated code. A bit map is employed to indicate the data that is absolute code and the data that is relocation data. The format of an object text record is:



Field	Size (Bytes)	Contents
3	1	Record type (ASCII $33).
Map	4	Bit map - each bit corresponds to one 16-bit word of absolute code or one set of relocatable data: 0 - Absolute code (16 bits each - word) 1 - Relocation data (1 to 12 bits)
ESDID	1	ESD index indicating the ESD for the section in which data from this record is to be placed.
Data	varies	Absolute code alternating with relocation data as per the bit map.

ESDID is a 1-byte ESD index that indicates in what section the code generated from this object text record is to be located. The way it works is:

A "program counter" is maintained for each section (relocatable, common, and absolute) in the relocatable object module being processed. Each program counter is initialized to the starting address of the section it represents. When an object text record is processed, the code generated from the record is placed at the address indicated by the program counter for the section whose ESD has the index indicated by ESDID. As code is generated, the program counter for the section into which the code is being placed is updated to indicate where the next code for that section should go.

Data is a variable length field that can contain up to 32 16-bit words of absolute code (code that does not need to be relocated), or up to 32 sets of relocation data, or any combination thereof. The data field is interpreted as:

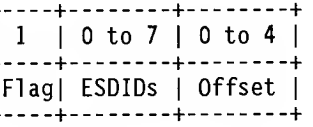
The highest order (leftmost) bit in the bit map corresponds to the first (leftmost) element in the data field. If the highest order bit in the bit map is a zero then the first 16-bit word in the data field is absolute code. However, if the highest order bit in the bit map is a 1, then the

A

first data item in the data field is relocation data. The second highest order bit in the bit map corresponds to the next data item in the data record in the same way, and so on. This processing of data proceeds until data corresponding to all 32 bits in the bit map has been processed or the end of the object text record (as indicated by the record length) is encountered, whichever comes first.

As previously mentioned, a zero bit in the bit map indicates one 16-bit word of absolute code that does not need to be relocated. This word is stored in the data field in two bytes in the exact form it is to appear in the resulting load module (or relocatable object module).

On the other hand, a 1 bit in the bit map indicates a set of relocation data in the data field. A set of relocation data is the data required to relocate a single 16-bit or a single 32-bit quantity. A set of relocation data is of variable length and can occupy from 1 to 12 bytes of data in the data field. The format of relocation data is:



Field	Size (Bytes)	Contents
Flag	1	Indicates type of relocation data: bits 7-5 - Number of ESDIDs (0-7) bit 4 - Reserved - must be 0 bit 3 - Size of relocated data 0 - 1 word (16 bits) 1 - 2 words (32 bits) bits 2-0 - Offset field length in bytes (0-4)
ESDIDs	0 to 7	ESD indices involved in relocation (1 byte each).
Offset	0 to 4	Constant offset involved in relocation.

The purpose of relocation data is to instruct the linkage editor how to calculate a relocated value. For each set of relocation data, this relocated value is initialized to zero. The relocation data is interpreted as:

Relocation data contains up to seven ESDIDs. Each ESDID in relocation data represents a numerical value. Typically, an ESDID will be the index of an ESD representing an external symbol reference. In that case, the numerical value associated with the ESDID is the address in the result module of the referenced symbol. However, an ESDID in relocation data may also be the index of an ESD that represents a section (relocatable, common, or absolute). Here, the numerical value associated with the ESDID

is the starting address of the particular section in the result module. The numerical values associated with the first, third, fifth, and seventh ESDIDs in relocation data are added to the relocated value while the values associated with the second, fourth, and sixth ESDIDs are subtracted from the relocated value. A zero ESDID in relocation data indicates that there is no ESDID for that position and therefore, no corresponding numerical value to be added or subtracted.

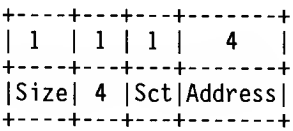
Relocation data may also contain a constant offset that is added to the relocated value. This offset may be from 0 to 4 bytes long and is always interpreted as a 2's complement value (thus the value -1 may be represented in one byte as $FF). An offset that is 0 bytes long indicates that there is no constant offset.

Once a relocated value has been calculated by adding and subtracting the proper ESDID values and adding the constant offset, it is placed in the resulting module in the size indicated by bit 3 of the flag byte. If this bit is off (0) then the relocated value will be put into the result module as a one-word 16-bit) value. If the bit is on (1), the relocated value will be placed into the result module as a two-word (32-bit) value.

Finally, if bits 7-5 of the flag byte indicate that there are no ESDIDs in the relocation data, then there must be an offset. Here, the offset (from 1 to 4 bytes long) is taken as a 2's complement value to be added to the current program counter for the section in which code from this object text record is being placed. Later code for the same section will be placed starting at the new location. This allows the relocating of program counters backward and forward for overwriting code that has already been generated ("fix-ups").

End Record (Type 4)

This record indicates the end of a relocatable object module and must be the last record in every module. It also contains information about the starting execution address of the module. The format of the end record is:



Field	Size (Bytes)	Contents
4	1	Record type (ASCII $34).
Sct	1	Section in which execution starts: 0-15 - Relocatable section 16 - Absolute section 17 - No starting address
Address	4	Starting execution address.

If the value of Sct is between 0 and 15, inclusive, then Address is interpreted as being relative to the start of the section. If the value of Sct is 16, then Address is an absolute address. Finally, if the value of Sct is 17, then no starting execution address has been specified for this module and address does not appear in the end record.

APPENDIX B

LOAD MODULE FILE FORMAT

Load Module Storage Format

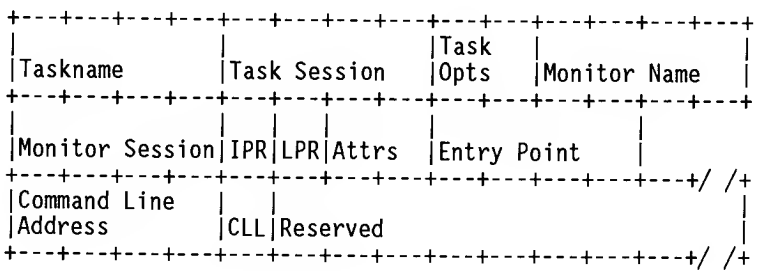
The VERSAdos operating system stores load modules in contiguous files. Each load module consists of a header block followed by a variable number of memory image blocks. Each block is 256 bytes long.

Loader Information Block

The first 256-byte block in a load module is called the Loader Information Block (LIB). Sometimes it is called the header block. The LIB contains all the necessary information about the load module except the actual data. The LIB consists of three major sections: the header, the segment allocation descriptors, and the memory image descriptors.

Header Section

The header part of the LIB occupies the first 48 bytes of the LIB and contains information about the task. The information is created when the load module is loaded into memory by the VERSAdos loader. The format is:



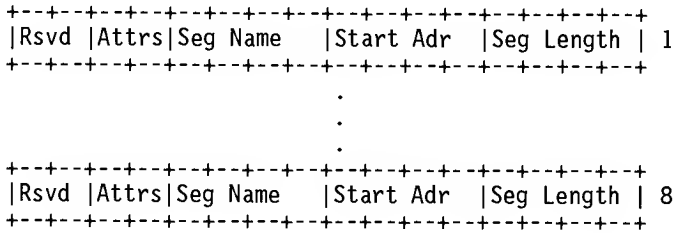
B

Field	Size (Bytes)	Offset	Contents
Taskname	4	$0	Name of task created when this module is loaded (ASCII).
Task Session	4	$4	Session number of task created when this module is loaded.
Task Opts	2	$8	Task options. Indicate desired options on bits. Bit values: 15 - Specifies monitor task 14 - Propagate monitor 13-0 - Not used (reserved)
Monitor Name	4	$A	Monitor taskname of this task (ASCII).
Monitor Session	4	$E	Session number of monitor task of this task.
IPR	1	$12	Initial priority of this task (0-255).
LPR	1	$13	Limit priority of this task (0-255).
Task Attrs	2	$14	Task attributes. Indicate desired attributes on bits. Bit values: 15 - This is a system task 14-13 - Not used (reserved) 12 - Dump task if aborted 11 - This task is position independent 10 - Assign the file from which this task is loaded to logical unit 8 9-8 - Not used (reserved) 7 - This task is real-time 6-0 - Not used (reserved)
Entry Point	4	$16	Beginning execution address of task.
Command Line Address	4	$1A	Address where command line is to be stored when loaded ($FFFFFFFF indicates don't store command line).
CLL	1	$1E	Command line length. Maximum number of characters; 1 to move to command line address.
Reserved	17	$1F	Not used (reserved).

B

Segment Allocation Descriptors

Immediately following the header section of the LIB are the Segment Allocation Descriptors (SADs). There are eight SADs and each one occupies 16 bytes. This makes for a total of 128 bytes. The SADs occupy the 49th ($30) through 176th ($AF) bytes in the LIB. Each SAD describes a Memory Management Unit (MMU) segment that is set up when the module is loaded. Currently, a task may have a maximum of four MMU segments allocated to it. However, the LIB allows for eight SADs for future expansion. The format of the SADs is:



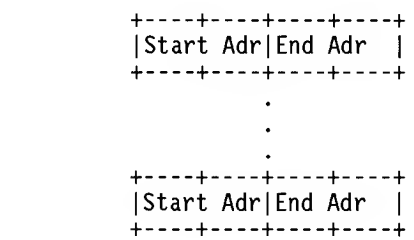
Field	Size (Bytes)	Contents
Rsvd	2	Not used (reserved).
Attrs	2	Segment attributes. Indicate desired attributes on bits. Bit values: 15 - Segment is to be allocated 14 - Segment is read only 13 - Segment is locally shareable 12 - Segment is globally shareable 11-0 - Not used (reserved)
Seg Name	4	Segment name (ASCII).
Start Adr	4	Logical starting address of segment.
Seg Length	4	Segment length (in bytes).

When the module is loaded, only those SADs that have bit 15 of the segment attributes field on are used to allocate segments. SADs with bit 15 of the segment attributes field off are ignored. Also, all the SADs with bit 15 on must be first in the list of SADs. In other words, the first SAD encountered with bit 15 of the segment attributes field off flags the end of the segments that must be allocated for this module. Space is always allocated for a full eight SADs in the LIB no matter how many segments are actually allocated for a given module.

B

Memory Image Descriptors

Immediately after the segment allocation descriptors in the loader information block are the Memory Image Descriptors (MIDs). There are 20 MIDs in a LIB and each MID occupies 4 bytes, which makes for a total of 80 bytes. The MIDs occupy the 177th ($B1) through 256th ($100) bytes of the LIB. Each MID defines a logical address space in memory into which data in the load module is to be located. The format of the MIDs is:



Field	Length (Bytes)	Contents
Start Adr	2	16 most significant bits of logical starting address of memory image (low order 8 bits assumed = $00).
End Adr	2	16 most significant bits of logical ending address of memory image (low order 8 bits assumed = $FF).

For each MID, there is a contiguous block of data in the memory image blocks of the load module that corresponds to the address space defined by the MID. The data corresponding to the MIDs appears in the load module in the order in which the MIDs appear. In other words, the data corresponding to the first MID is first in the load module, with the data corresponding to the second MID immediately following, and so on. Processing is done on all twenty of the MIDs or until one is encountered with a starting address of $000000 and an ending address of $FFFFFF, whichever occurs first.

Memory Image Blocks

Immediately following the LIB in a load module are a variable number of contiguous memory image blocks. Each memory image block contains 256 bytes of code/data that is to be loaded into memory, without alteration, when the task is loaded. The number of memory image blocks in a load module depends on the number of MIDs in the LIB and the address spaces defined by them. MIDs define memory images in multiples of pages (256 bytes). Therefore, all the data in any given memory image block belongs to one and only one MID.

NOTE: When generating a file that is to be booted directly from the firmware monitor and not invoked by VERSAdos, the following rules are applied:

a. The program must fit into one segment, e.g., SEG SEG0:0-15 $0.

b. The program should start from address zero with the first longword being the beginning stack pointer and the second longword being the address where execution is to begin.

c. The entry point of the program should be address 0, the address of the beginning stack pointer mentioned above.

THIS PAGE INTENTIONALLY LEFT BLANK.

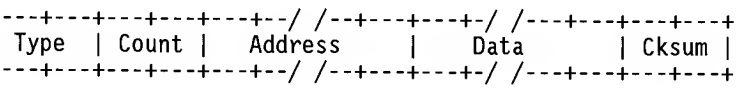
APPENDIX C

S-RECORD FILE FORMAT

An S-record file consists of a sequence of specially formatted ASCII character strings. Several fields within these records have groups of characters that must be interpreted as hexadecimal values of one to four bytes in length. An S-record will be less than or equal to 70 bytes in length. Since each S-record requires 10 to 14 bytes in fixed overhead for the type, byte count, address and checksum fields, the variable length data field may be allocated up to 60 bytes. This translates to 60 characters or 30 character pairs or bytes of data per data record from the user viewpoint.

The S-record file output by the linker is not in any particular order so the order of S-records within a file is of no significance.

The general format of an S-record is:



Field	Size (Bytes)	Contents
Type	2	ASCII bytes, whose associated characters describe the type of record (S0, S1, S2, S3, S7, S8, or S9).
Count	2	ASCII bytes whose associated characters, when paired and interpreted as a byte value, display the count of the remaining character pairs in the record.
Address	4-8	ASCII bytes whose associated characters, when paired and interpreted as a two to four byte value, display the address where the data field is to be loaded into memory.
Data	0-60	ASCII bytes whose associated characters, when paired and interpreted as byte values, represent memory loadable data or descriptive information.
Cksum	2	(Checksum) ASCII bytes whose associated characters, when paired and interpreted as a byte value, display the least significant byte of the one's complement of the sum of the byte values represented by the pairs of ASCII characters making up the count, the address, and the data fields.

C

The "S0" Record

The type of record field is "S0" ($5330). The address field is unused and filled with zeros ($30303030). The user supplies the header information in the data field with the interactive user command **IDENT**. The subfields are:

Subfield	Size (Bytes)	Contents
mname	20	module name
ver	2	version number
rev	2	revision number
description	0-36	text comment

Each of the subfields is composed of ASCII bytes whose associated characters, when paired, represent one byte hexadecimal values for the version and revision numbers, or the hexadecimal values of the ASCII characters comprising the module name and description specified with the **IDENT** command.

If the **IDENT** command is not used, the filename portion of the output file the linker is creating is used as the module name; the version and revision numbers are 1; and there is no description.

The "S1" Record

The type of record field is "S1" ($5331). The address field is interpreted as a 2-byte address. The data field is composed of memory loadable data.

The "S2" Record

The type of record field is "S2" ($5332). The address field is interpreted as a 3-byte address. The data field is composed of memory loadable data.

The "S3" Record

The type of record field is "S3" ($5333). The address field is interpreted as a 4-byte address. The data field is composed of memory loadable data.

The "S7", "S8", and "S9" Records

The type of record field is "S7", "S8", and "S9" ($5337, $5338, and $5339), respectively. The address field contains the starting execution address specified by the user with the interactive user command **ENTRY**. The first entry point encountered in the object module's input is used, if an **ENTRY** command is not specified. If no starting address is encountered, the

beginning address of the first segment is used. If none of these methods is used to specify the starting address, this field is set to zeros. The address field of the "S7", "S8", and "S9" records is four, three, and two bytes, respectively. There is no data field.

C

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX D

DEBUG FILE FORMAT

The format of debug files is similar to that of relocatable object module files in that they are stored in sequential files with fixed length records of 256 bytes. Records not completely filled with information are padded with $FF to fill 256 bytes.

Debug File Contents

A debug file contains information taken from three sources: the relocatable object modules used as input to the linker, their associated .RS files (if they exist), and the load module information block. This information is organized into a .DB header record and one or more additional records per relocatable object module included in the link. The module information is organized into a Module Header Record, zero or more Module Symbol Records, and zero or more Module Index Records per module.

DB Header Record

FIELD	SIZE (BYTES)	CONTENTS
version number	4	$00000001
filename	8	First eight characters of the first relocatable object module included in the link.
# modules	2	The count of the relocatable object modules included in the link.
filler	2	Two bytes of $FF.
seg0 name . . . seg7 name	4 4	Up to eight segment names, taken from the load module information block.

Module Header Record

FIELD	SIZE (BYTES)	CONTENTS
filler	1	One byte of $00.
A	1	The ASCII character "A".
name	8	The first eight characters of the relocatable object module name.
# symbols	2	The count of the symbols associated with the module, from the .RS file.
filler	2	Two bytes of $00.
next rec	2	The record number associated with the module header record of the next module; this field will contain zeros in the last module header record.
sec0 seg#	1	The number of the segment where the section resides.
sec0 addr	3	The logical address where the section resides.
.		
.		
.		
sec16 seg#	1	
sec16 addr	3	

Module Symbol Records

Each of these records contains information associated with up to 16 symbols. This information is composed of the following fields:

FIELD	SIZE (BYTES)	CONTENTS
name	8	The first eight characters of the symbol name.
length	2	A byte of $00 followed by a byte of $04.
attr	1	An attribute of the symbol: C - Named common E - EQU symbol L - Label symbol S - SET symbol
section	1	The section where the symbol was defined.
value	4	An address or value associated with the symbol.

Module Index Records

The module symbol records contain the first four characters of up to 64 symbol names. These records are used as indices into the module symbol records.

APPENDIX E

EXAMPLES

The following are examples of the various types of files created by the linkage editor. First are the assembly language sources of three separate relocatable object modules. Along with each source is a dump of the relocatable object module that was created when the source was assembled. Next is the listing created by the linkage editor when the three modules were linked. Following this is a dump of the load module created by the link. Next is a dump of the debug file that was also created by the link. A second linkage editor listing shows the creation of an S-record module; it is followed by a listing of an S-record module.

Source of First Relocatable Object Module

```
MODULE1   IDNT   1,0 File format example module one
*
          XREF   LABEL3              XREF to anywhere
          XDEF   LABEL1              XDEF in relocatable section
          XDEF   LABEL2              XDEF in absolute section
*
          SECTION.S 1                Short address section
          DC.B   'Start of section one in module one'
LABEL1    DS.B   100                 Will cause relocation of PC
          DC.B   'End of section one in module one'
*
COMMON1   SECTION 2                  Common section
          DC.B   'Common section one in module one'
*
          ORG    $2000               Absolute section
LABEL2    DC.B   'Absolute section in module one'
          MOVE.W LABEL3,D0           Will need relocation data
          MOVE.B LABEL1,D0           Will need relocation data
*
          SECTION 3                  Regular section
          DC.B   'Section three in module one'
          MOVE.W LABEL1,D0           Will need relocation data
          COMLINE 80                 Command line in relocatable section
*
          END    LABEL2              Start address in absolute section
```

Dump of First Relocatable Object Module

```
    SN=$0          0
00    4A 31 4D 4F 44 55 4C 45  31 20 20 20 01 00 41 46    J1MODULE1   ..AF
10    49 58 20 00 01 45 58 41  4D 50 4C 45 20 4D 4F 44    IX ..EXAMPLE MOD
20    55 4C 45 31 20 53 41 16  08 03 05 11 81 46 69 6C    ULE1 SA......Fil
30    65 20 66 6F 72 6D 61 74  20 65 78 61 6D 70 6C 65    e format example
40    20 6D 6F 64 75 6C 65 20  6F 6E 65 52 32 41 4C 41     module oneR2ALA
50    42 45 4C 31 20 20 20 20  00 00 00 22 50 4C 41 42    BEL1    ..."PLAB
60    45 4C 32 20 20 20 20 00  00 20 00 70 4C 41 42 45    EL2     .. .pLABE
70    4C 33 20 20 20 20 31 00  00 00 A6 23 00 00 00 70    L3    1....#...p
80    00 00 00 00 28 00 00 20  00 12 43 4F 4D 4D 4F 4E    ....(.. ..COMMON
90    31 20 20 20 00 00 00 20  83 00 00 00 20 4F 46 33    1   ... .... OF3
A0    00 00 40 00 02 53 74 61  72 74 20 6F 66 20 73 65    ..@..Start of se
B0    63 74 69 6F 6E 20 6F 6E  65 20 69 6E 20 6D 6F 64    ction one in mod
C0    75 6C 65 20 6F 6E 65 01  64 45 6E 64 20 6F 66 20    ule one.dEnd of
D0    73 65 63 74 69 6F 6E 20  6F 6E 65 20 69 6E 20 6D    section one in m
E0    6F 64 75 6C 65 0A 33 00  00 00 00 02 20 6F 6E 65    odule.3..... one
F0    26 33 00 00 00 00 13 43  6F 6D 6D 6F 6E 20 73 65    &3.....Common se

    SN=$1          1
00    63 74 69 6F 6E 20 6F 6E  65 20 69 6E 20 6D 6F 64    ction one in mod
10    75 6C 65 20 6F 6E 65 2D  33 00 00 A0 00 12 41 62    ule one-3.....Ab
20    73 6F 6C 75 74 65 20 73  65 63 74 69 6F 6E 20 69    solute section i
30    6E 20 6D 6F 64 75 6C 65  20 6F 6E 65 30 39 28 11    n module one09(.
40    10 38 21 02 22 29 33 00  01 80 00 04 53 65 63 74    .8!.")3.....Sect
50    69 6F 6E 20 74 68 72 65  65 20 69 6E 20 6D 6F 64    ion three in mod
60    75 6C 65 20 6F 6E 65 00  30 38 21 02 22 01 50 06    ule one.08!.".P.
70    34 10 00 00 20 00 00 00  00 00 00 00 00 00 00 00    4... ............
80    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
90    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
A0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
B0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
C0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
D0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
E0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
F0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
```

E

Source of Second Relocatable Object Module

```
MODULE2   IDNT   1,0 File format example module two
*
          XREF.S 1:LABEL1          XREF to particular section
*
          SECTION.S 1              Short address section
          MOVE.W LABEL1,D0         Will need relocation data
START     DC.B   'Section one in module two'
*
          ORG    $2110             Absolute section
          DC.B   'Absolute section in module two'
          COMLINE 160              Command line in absolute section
          MOVE.W LABEL1,D0         Will need relocation data
*
COMMON1   SECTION 2                Common section
          DS.B   200
*
COMMON2   SECTION 5                Common Section
          DC.B   'Common section two in module two'
*
          END    START             Start address in relocatable section
```

Dump of Second Relocatable Object Module

```
     SN=$0          0
00     4A 31 4D 4F 44 55 4C 45   32 20 20 20 01 00 41 46    J1MODULE2   ..AF
10     49 58 20 00 01 45 58 41   4D 50 4C 45 20 4D 4F 44    IX ..EXAMPLE MOD
20     55 4C 45 32 20 53 41 08   36 39 05 12 81 46 69 6C    ULE2 SA.69...Fil
30     65 20 66 6F 72 6D 61 74   20 65 78 61 6D 70 6C 65    e format example
40     20 6D 6F 64 75 6C 65 20   74 77 6F 3E 32 61 4C 41     module two>2aLA
50     42 45 4C 31 20 20 20 20   31 00 00 00 1E 00 00 00    BEL1    1.......
60     00 C2 00 00 21 10 12 43   4F 4D 4D 4F 4E 31 20 20    ....!..COMMON1
70     20 00 00 00 C8 15 43 4F   4D 4D 4F 4E 32 20 20 20     .....COMMON2
80     00 00 00 20 90 00 00 21   2E 9F 24 33 40 00 00 00    ... ...!..$3@...
90     02 30 38 20 11 53 65 63   74 69 6F 6E 20 6F 6E 65    .08 .Section one
A0     20 69 6E 20 6D 6F 64 75   6C 65 20 74 77 6F 00 2B     in module two.+
B0     33 00 01 40 00 12 41 62   73 6F 6C 75 74 65 20 73    3..@..Absolute s
C0     65 63 74 69 6F 6E 20 69   6E 20 6D 6F 64 75 6C 65    ection in module
D0     20 74 77 6F 02 00 A0 30   38 20 11 09 33 80 00 00     two...08 ..3...
E0     00 13 02 00 C8 26 33 00   00 00 00 14 43 6F 6D 6D    .....&3.....Comm
F0     6F 6E 20 73 65 63 74 69   6F 6E 20 74 77 6F 20 69    on section two i

     SN=$1          1
00     6E 20 6D 6F 64 75 6C 65   20 74 77 6F 06 34 01 00    n module two.4..
10     00 00 04 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
20     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
30     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
40     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
50     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
60     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
70     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
80     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
90     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
A0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
B0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
C0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
D0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
E0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
F0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ................
```

E

Source of Third Relocatable Object Module

```
MODULE3   IDNT   1,0 File format example module three
*
          XREF   LABEL2              XREF to any section
          XDEF   LABEL3              XDEF in relocatable section
*
COMMON2   SECTION 5                  Common section
          COMLINE 10                 Command line in common section
*
COMMON1   SECTION 2                  Common section
          DC.B   'Common section two in module three'
*
          SECTION.S 1                Short address section
          DC.B   'Section one in module three'
*
          SECTION 5                  Regular section
          DC.B   'Section five in module three'
LABEL3    MOVE.L LABEL2,D0           Will need relocation data
*
          END
```

Dump of Third Relocatable Object Module

```
     SN=$0          0
00     4D 31 4D 4F 44 55 4C 45  33 20 20 20 01 00 41 46    M1MODULE3   ..AF
10     49 58 20 00 01 45 58 41  4D 50 4C 45 20 4D 4F 44    IX ..EXAMPLE MOD
20     55 4C 45 33 20 53 41 08  32 22 05 12 81 46 69 6C    ULE3 SA.2"...Fil
30     65 20 66 6F 72 6D 61 74  20 65 78 61 6D 70 6C 65    e format example
40     20 6D 6F 64 75 6C 65 20  74 68 72 65 65 27 53 32     module three'S2
50     70 4C 41 42 45 4C 32 20  20 20 20 45 4C 41 42 45    pLABEL2    ELABE
60     4C 33 20 20 20 20 00 00  00 1C 31 00 00 00 1C 25    L3    ....1....%
70     00 00 00 22 15 43 4F 4D  4D 4F 4E 32 20 20 20 00    ...".COMMON2   .
80     00 00 0A 12 43 4F 4D 4D  4F 4E 31 20 20 20 00 00    ....COMMON1   ..
90     00 22 A5 43 4F 4D 4D 4F  4E 32 20 20 20 00 00 00    .".COMMON2    ...
A0     00 09 08 33 80 00 00 00  12 01 0A 28 33 00 00 00    ...3.......(3...
B0     00 13 43 6F 6D 6D 6F 6E  20 73 65 63 74 69 6F 6E    ..Common section
C0     20 74 77 6F 20 69 6E 20  6D 6F 64 75 6C 65 20 74     two in module t
D0     68 72 65 65 22 33 00 00  00 00 02 53 65 63 74 69    hree"3.....Secti
E0     6F 6E 20 6F 6E 65 20 69  6E 20 6D 6F 64 75 6C 65    on one in module
F0     20 74 68 72 65 65 00 26  33 00 01 00 00 06 53 65     three.&3.....Se

     SN=$1          1
00     63 74 69 6F 6E 20 66 69  76 65 20 69 6E 20 6D 6F    ction five in mo
10     64 75 6C 65 20 74 68 72  65 65 20 39 28 11 02 34    dule three 9(..4
20     11 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
30     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
40     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
50     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
60     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
70     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
80     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
90     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
A0     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
B0     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
C0     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
D0     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
E0     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
F0     00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
```

E

Linkage Editor Listing

```
Command Line:

LINK ,EXAMPLE.LOADMOD.LO,EXAMPLE.LINKMAP.LL;IXHMD


Options in Effect:  A,-B,D,H,I,-L,M,O,P,-Q,-R,-S,-U,-W,X


User Commands:

SEG SEG1(RLG):1-4
SEG SEG2:5
SEG SEG3(R):14 $2000
IN EXAMPLE.MODULE1.RO,EXAMPLE.MODULE2.RO,EXAMPLE.MODULE3.RO
TASK TEST '$1234
MONITOR MON $1234
PRIORITIES 10,100
OPTIONS M
ATTRIBUTES SD
END


Object Module Header Information:

Module      Ver Rev Language Date     Time     Creation Filename

MODULE1       1   0 Assembly 05/11/81 16:08:03 FIX:1.EXAMPLE.MODULE1.SA
            File format example module one

MODULE2       1   0 Assembly 05/12/81 08:36:39 FIX:1.EXAMPLE.MODULE2.SA
            File format example module two

MODULE3       1   0 Assembly 05/12/81 08:32:22 FIX:1.EXAMPLE.MODULE3.SA
            File format example module three'


Load Map:


Segment SEG1(R,L,G): 00000000 000002FF 1,2,3,4
Module       S   T  Start     End       Externally Defined Symbols

MODULE1      1   S  00000000  000000A5  LABEL1     00000022
MODULE2      1   S  000000A6  000000C3
MODULE3      1   S  000000C4  000000DF
COMMON1      2   C  000000E0  000001A7
MODULE1      3      000001A8  00000217
```

```
Segment SEG2: 00000300 000003FF 5
Module      S   T  Start     End       Externally Defined Symbols

MODULE3     5      00000300  00000321  LABEL3     0000031C
COMMON2     5  C   00000322  00000341


Segment SEG3(R): 00002000 000021FF 14
Module      S   T  Start     End       Externally Defined Symbols

MODULE1         A  00002000  00002027  LABEL2     00002000
MODULE2         A  00002110  000021D1
```

Table of Externally Defined Symbols:

```
Name        Address   Module     Displ     Sect Seg   Library      Input

LABEL1      00000022  MODULE1    00000022   1   SEG1               MODULE1 .RO
LABEL2      00002000  MODULE1                   SEG3               MODULE1 .RO
LABEL3      0000031C  MODULE3    0000001C   5   SEG2               MODULE3 .RO
```

```
Unresolved References: None


Multiply Defined Symbols: None


Lengths (in bytes):

     Segment      Hex          Decimal

      SEG1     00000300            768
      SEG2     00000100            256
      SEG3     00000200            512
Total Length   00000600           1536


   No Errors
   No Warnings


Load module :EXAMPLE.LOADMOD.LO has been created.
```

E

Dump of Load Module

```
    SN=$0          0
00    54 45 53 54 31 32 33 34  80 00 4D 4F 4E 20 00 00   TEST1234..MON ..
10    12 34 0A 64 90 00 00 00  20 00 00 00 01 C8 4F 00   .4.d.... .....O.
20    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
30    00 00 F0 00 53 45 47 31  00 00 00 00 00 00 03 00   ....SEG1........
40    00 00 80 00 53 45 47 32  00 00 03 00 00 00 01 00   ....SEG2........
50    00 00 C0 00 53 45 47 33  00 00 20 00 00 00 02 00   ....SEG3.. .....
60    00 00 00 00 00 00 00 00  00 00 00 00 FF FF FF FF   ................
70    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
80    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
90    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
A0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
B0    00 00 00 02 00 03 00 03  00 20 00 21 00 00 FF FF   ......... .!....
C0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
D0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
E0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
F0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................

    SN=$1          1
00    53 74 61 72 74 20 6F 66  20 73 65 63 74 69 6F 6E   Start of section
10    20 6F 6E 65 20 69 6E 20  6D 6F 64 75 6C 65 20 6F    one in module o
20    6E 65 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ne..............
30    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
40    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
50    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
60    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
70    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
80    00 00 00 00 00 00 45 6E  64 20 6F 66 20 73 65 63   ......End of sec
90    74 69 6F 6E 20 6F 6E 65  20 69 6E 20 6D 6F 64 75   tion one in modu
A0    6C 65 20 6F 6E 65 30 38  00 22 53 65 63 74 69 6F   le one08."Sectio
B0    6E 20 6F 6E 65 20 69 6E  20 6D 6F 64 75 6C 65 20   n one in module
C0    74 77 6F 00 53 65 63 74  69 6F 6E 20 6F 6E 65 20   two.Section one
D0    69 6E 20 6D 6F 64 75 6C  65 20 74 68 72 65 65 00   in module three.
E0    43 6F 6D 6D 6F 6E 20 73  65 63 74 69 6F 6E 20 74   Common section t
F0    77 6F 20 69 6E 20 6D 6F  64 75 6C 65 20 74 68 72   wo in module thr

    SN=$2          2
00    65 65 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ee..............
10    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
20    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
30    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
40    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
50    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
60    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
70    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
80    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
90    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
A0    00 00 00 00 00 00 00 00  53 65 63 74 69 6F 6E 20   ........Section
B0    74 68 72 65 65 20 69 6E  20 6D 6F 64 75 6C 65 20   three in module
C0    6F 6E 65 00 30 38 00 22  00 00 00 00 00 00 00 00   one.08."........
D0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
E0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
F0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
```

```
     SN=$3          3
00     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
10     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
20     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
30     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
40     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
50     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
60     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
70     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
80     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
90     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
A0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
B0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
C0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
D0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
E0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
F0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................

     SN=$4          4
00     53 65 63 74 69 6F 6E 20   66 69 76 65 20 69 6E 20   Section five in
10     6D 6F 64 75 6C 65 20 74   68 72 65 65 20 39 00 00   module three 9..
20     20 00 43 6F 6D 6D 6F 6E   20 73 65 63 74 69 6F 6E    .Common section
30     20 74 77 6F 20 69 6E 20   6D 6F 64 75 6C 65 20 74    two in module t
40     77 6F 00 00 00 00 00 00   00 00 00 00 00 00 00 00   wo..............
50     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
60     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
70     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
80     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
90     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
A0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
B0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
C0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
D0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
E0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
F0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................

     SN=$5          5
00     41 62 73 6F 6C 75 74 65   20 73 65 63 74 69 6F 6E   Absolute section
10     20 69 6E 20 6D 6F 64 75   6C 65 20 6F 6E 65 30 39    in module one09
20     00 00 03 1C 10 38 00 22   00 00 00 00 00 00 00 00   .....8."........
30     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
40     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
50     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
60     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
70     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
80     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
90     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
A0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
B0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
C0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
D0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
E0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
F0     00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
```

E

```
    SN=$6          6
00    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
10    41 62 73 6F 6C 75 74 65  20 73 65 63 74 69 6F 6E    Absolute section
20    20 69 6E 20 6D 6F 64 75  6C 65 20 74 77 6F 00 00     in module two..
30    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
40    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
50    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
60    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
70    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
80    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
90    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
A0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
B0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
C0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 30 38    ..............08
D0    00 22 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ."..............
E0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
F0    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
```

Dump of Debug File

```
    SN=$0          0
00    00 00 00 01 4D 4F 44 55  4C 45 31 20 00 03 FF FF    ....MODULE1 ....
10    53 45 47 31 53 45 47 32  53 45 47 33 00 00 00 00    SEG1SEG2SEG3....
20    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ................
30    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
50    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
60    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................

    SN=$1          1
00    00 41 4D 4F 44 55 4C 45  31 20 00 03 00 00 00 04    .AMODULE1 ......
10    FF FF FF FF 00 00 00 00  FF FF FF FF 00 00 01 AB    ................
20    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
30    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
50    02 00 20 00 FF FF FF FF  FF FF FF FF FF FF FF FF    ................
60    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................

    SN=$2          2
00    43 4F 4D 4D 4F 4E 31 20  00 04 43 02 00 00 00 E0    COMMON1 ..C.....
10    4C 41 42 45 4C 31 20 20  00 04 4C 01 00 00 00 22    LABEL1  ..L....."
20    4C 41 42 45 4C 32 20 20  00 04 4C 10 00 00 20 00    LABEL2  ..L.....
30    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
50    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
60    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
```

E

```
    SN=$3          3
00    4C 41 42 45 FF FF FF FF   FF FF FF FF FF FF FF FF    LABE............
10    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
20    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
30    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
50    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
60    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................

    SN=$4          4
00    00 41 4D 4F 44 55 4C 45   32 20 00 03 00 00 00 07    .AMODULE2 ......
10    FF FF FF FF 00 00 00 A6   FF FF FF FF FF FF FF FF    ................
20    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
30    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
50    02 00 21 10 FF FF FF FF   FF FF FF FF FF FF FF FF    ..!.............
60    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................

    SN=$5          5
00    43 4F 4D 4D 4F 4E 31 20   00 04 43 02 00 00 00 E0    COMMON1 ..C.....
10    43 4F 4D 4D 4F 4E 32 20   00 04 43 05 00 00 03 22    COMMON2 ..C....."
20    53 54 41 52 54 20 20 20   00 04 4C 01 00 00 00 04    START   ..L.....
30    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
50    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
60    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
```

```
    SN=$6          6
00    53 54 41 52 FF FF FF FF   FF FF FF FF FF FF FF FF    STAR............
10    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
20    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
30    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
50    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
60    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................

    SN=$7          7
00    00 41 4D 4F 44 55 4C 45   33 20 00 03 00 00 00 00    .AMODULE3 ......
10    FF FF FF FF 00 00 00 C4   FF FF FF FF FF FF FF FF    ................
20    FF FF FF FF 01 00 03 00   FF FF FF FF FF FF FF FF    ................
30    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
50    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
60    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................

    SN=$8          8
00    43 4F 4D 4D 4F 4E 31 20   00 04 43 02 00 00 00 E0    COMMON1 ..C.....
10    43 4F 4D 4D 4F 4E 32 20   00 04 43 05 00 00 03 22    COMMON2 ..C....".
20    4C 41 42 45 4C 33 20 20   00 04 4C 05 00 00 00 1C    LABEL3  ..L.....
30    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
40    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
50    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
60    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
70    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
80    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
90    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
A0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
B0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
C0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
D0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
E0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
F0    FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF    ................
```

E

```
     SN=$9          9
00     4C 41 42 45 FF FF FF FF  FF FF FF FF FF FF FF FF    LABE............
10     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
20     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
30     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
40     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
50     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
60     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
70     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
80     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
90     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
A0     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
B0     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
C0     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
D0     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
E0     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
F0     FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF    ................
```

E

Second Linkage Editor Listing

```
Command Line:

LINK ,EXAMPLE.SRECMOD.MX,EXAMPLE.LINKMAP.LL;IXHMDQ


Options in Effect:  A,-B,D,H,I,-L,M,-O,P,Q,-R,-S,-U,-W,X


User Commands:

SEG SEG1(RLG):1-4
SEG SEG2:5
SEG SEG3(R):14 $2000
IN EXAMPLE.MODULE1.RO,EXAMPLE.MODULE2.RO,EXAMPLE.MODULE3.RO
QUIT


Object Module Header Information:

Module      Ver Rev Language Date     Time     Creation Filename

MODULE1       1   0 Assembly 12/15/81 13:23:23 FIX:7.EXAMPLE.MODULE1.SA
            File format example module one

MODULE2       1   0 Assembly 12/15/81 13:31:15 FIX:7.EXAMPLE.MODULE2.SA
            File format example module two

MODULE3       1   0 Assembly 12/15/81 13:32:01 FIX:7.EXAMPLE.MODULE3.SA
            File format example module three'


Load Map:

Segment SEG1(R,L,G): 00000000 000002FF 1,2,3,4
Module      S   T  Start     End       Externally Defined Symbols

MODULE1       1  S  00000000  000000A5  LABEL1      00000022
MODULE2       1  S  000000A6  000000C3
MODULE3       1  S  000000C4  000000DF
COMMON1       2  C  000000E0  000001A7
MODULE1       3     000001A8  00000217


Segment SEG2: 00000300 000003FF 5
Module      S   T  Start     End       Externally Defined Symbols

MODULE3       5     00000300  00000321  LABEL3      0000031C
COMMON2       5  C  00000322  00000341
```

```
Segment SEG3(R): 00002000 000021FF 14
Module      S   T  Start     End       Externally Defined Symbols

MODULE1         A  00002000  00002027  LABEL2     00002000
MODULE2         A  00002110  000021D1


Table of Externally Defined Symbols:

Name        Address   Module      Displ     Sect Seg   Library      Input

LABEL1         00000022  MODULE1     00000022   1  SEG1              MODULE1
.RO
LABEL2         00002000  MODULE1                   SEG3              MODULE1
.RO
LABEL3         0000031C  MODULE3     0000001C   5  SEG2              MODULE3
.RO


Unresolved References: None


Multiply Defined Symbols: None


Lengths (in bytes):

     Segment     Hex         Decimal

      SEG1     00000300           768
      SEG2     00000100           256
      SEG3     00000200           512
Total Length   00000600          1536


   No Errors
   No Warnings


S-record module :EXAMPLE.SRECMOD.MX has been created.
```

Listing of S-Record Module

```
S00F000053524543D4F44202020010181
S1210000537461727420F662073656374696F6E206F6E6520696E206D6F64756C65C7
S107001E206F6E6578
S1210086456E64206F662073656374696F6E206F6E6520696E206D6F64756C65206FA9
S10500A46E6583
S12100E0436F6D6D6F6E2073656374696F6E206F6E6520696E206D6F64756C65206FF2
S10500FE6E6529
S12120004162736F6C75746520736563746974696F6E20696E206D6F64756C65206F6E656B
S10D201E30390000031C10380022C2
S12101A853656374696F6E2074687265652020696E206D6F64756C65206F6E65003038C1
S10501C6002211
S12100A6303800225365637469 6F6E206F6E6520696E206D6F64756C652074776F0060
S12121104162736F6C75746520736563746974696F6E20696E206D6F64756C652074776F42
S10721CE303800227F
S1210322436F6D6D6F6E2073656374696F6E2074776F20696E206D6F64756C65207490
S1050340776FD1
S12100E0436F6D6D6F6E2073656374696F6E2074776F20696E206D6F64756C652074D5
S10700FE6872656556
S11F00C453656374696F6E206F6E6520696E206D6F64756C652074687265650010
S1210300536563746974696F6E206669766520696E206D6F64756C6520746872656520390E
S107031E00002000B7
S9032000DC
```

E

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX F

ERROR/WARNING MESSAGES

During runtime, the linker may generate its own messages. This appendix lists them under ERROR MESSAGES and WARNING MESSAGES. However, runtime errors may occur from these sources as well, because the linker is written in Pascal and operates in a VERSAdos environment. A complete reference of Pascal and VERSAdos runtime errors is found in the VERSAdos Messages Reference Manual.

ERROR MESSAGES

The form of the error messages generated by the linkage editor are:

```
** ERROR nnn - description
```

where:

nnn is a three-digit error number.

description is a general description of the type of error.

Errors are divided into classes where each class has one type of error message. The first digit of the 3-digit error number determines the class. Thus, whenever an error occurs, its error number is printed along with the general error message for its class. The various error classes and their specific errors are:

F

Class 1 - Illegal Command Line

The general message for errors of this class is:

```
** ERROR 1nn - Illegal command line
```

This error shows there is something wrong with the command line used to invoke the linkage editor. If an error of this class occurs, the linkage editor does not continue and control is returned to the operating system. The errors for this type are:

133 No filename: No filename specification found while scanning the command line.

134 Illegal filename: A general syntax error was encountered while scanning a filename specification.

135 Illegal device name: A syntax error was encountered while scanning the device name field of a filename specification.

136 Illegal volume name: A syntax error was encountered while scanning the volume name field of a filename specification.

137 No user number: While scanning a filename specification, a user number was expected but not found.

138 Illegal user number: A syntax error was encountered while scanning the user number in a filename specification.

139 Illegal catalog name: A syntax error was encountered while scanning the catalog name in a filename specification.

140 Illegal extension: A syntax error was encountered while scanning the extension in a filename specification.

141 Illegal key(s): A syntax error was encountered while scanning the key(s) in a filename specification.

142 Filename already specified: The list of input files on the command line specified the same filename twice.

143 Illegal option specification: The option field of the command line contains a syntax error.

144 Option conflict: A conflict exists between two or more options specified on the command line. For example, two mutually exclusive options (**O** and **R**), were specified.

145 No output filename specified: The output filename was not specified on the command line. An output file must be specified on the command line when relocatable output is requested (via the **R** option).

147 Illegal option syntax for the **W** option: The command line contains a syntax error in the **W** option.

148 Option conflict: The bit width chosen for addressable memory conflicts with the type of output module chosen.

Class 2 - Illegal User Command Line

The general message for errors of this class is:

** ERROR 2nn - Illegal user command line

This error shows that there is an error in a user command line specified to the linkage editor. When an error of this class occurs, the offending user command is ignored and the user is prompted for another command. The errors of this class are:

200 Command line too long: The user command line specified is too long. The maximum length of a user command is 132 characters.

20I Illegal character: An illegal character was encountered while scanning the user command.

202 Illegal command verb: The user has specified an unknown command name.

203 Too many arguments: Too many arguments specified for this command.

204 Not enough arguments: Too few arguments given for this command.

206 No digits in number expected: Expected a number but it was not found when the user command was scanned.

207 Illegal number: An illegal digit was found while scanning a number. For example, found the digit "9" in an octal number.

208 Illegal section number: A section number specified was not between 0 to 15.

209 Illegal section number range: The specification of a range of section numbers was not of the proper form. For example, the first section number must be less than or equal to the second section number.

210 Section number already specified: The same section number was specified twice in a **START** or **SEGMENT** command.

2II Section number already assigned to a segment: A section number specified in a **SEGMENT** command has already been assigned to a different segment.

2I2 Section not assigned to a segment: A section specified in a **START** command has not been assigned to a segment.

2I3 Not all sections assigned to same segment: All the section numbers specified in a **START** command have not been assigned to the same segment.

2I4 No name: While scanning a user command, a symbol, module, segment, or taskname was expected but not found.

2I5 Illegal name: A syntax error was encountered while scanning a symbol, module, segment, or taskname. Symbol and module names must be from one to ten alphanumeric characters, with the first character being alpha; segment and tasknames must be from one to four alphanumeric characters, with alpha in the first character.

216 Undefined symbol: A symbol specified in the user command has not been encountered as an **XDEF** in the relocatable object modules input.

2I7 Command not legal for absolute output: This command cannot be used to create a load module.

218 Command not legal for relocatable output: This command cannot be specified when creating a relocatable object module.

219 All segments used: Made an attempt to create more than four segments with **SEGMENT** commands; four segments is the maximum.

220 Command not legal for S-record output: This command cannot be used to create an S-record module.

221 Segment does not exist: The segment specified in the command does not have any sections assigned to it and, therefore, does not exist.

222 No more **SEGMENT** commands allowed: No more **SEGMENT** commands can be specified.

223 Illegal segment start address: The address given as the starting address of a segment is not legal. The last byte of the starting address of a segment must be $00.

224 Illegal segment end address: The address given as the ending address of a segment is not legal. The last byte of the ending address of a segment must be $FF.

225 Conflicting address space: The address space specified for a segment conflicts with the address space previously specified for another segment.

226 Illegal address: A syntax error in an address was encountered while scanning the command line.

227 Address out of legal range: The address in a **START**, **ENTRY**, or **COMLINE** command is not in the range of the segment referred to by the command.

228 Illegal attribute specification: A syntax error in the attributes was encountered while scanning a **SEGMENT** command.

231 Module name already specified: The same module name was specified twice for the same file in an **INPUT** command.

232 Illegal command line length: The length specified in a **COMLINE** command is not between 1 to 256, inclusive.

233 No filename: While scanning the command line, a filename specification was expected but not found.

234 Illegal filename: A general syntax error was encountered while attempting to scan a file name specification.

235 Illegal device name: A syntax error in the device name field was encountered while scanning the filename specification.

236 Illegal volume name: A syntax error in the volume name field was encountered while scanning the filename specification.

237 No user number: While scanning a filename specification, a user number was expected but not found.

238 Illegal user number: A syntax error in the user number was encountered while scanning the filename specification.

239 Illegal catalog name: A syntax error in the catalog name was encountered while scanning the filename specification.

240 Illegal extension: A syntax error in the extension was encountered while scanning a filename specification.

241 Illegal key(s): A syntax error in the key(s) was encountered while scanning a filename specification.

242 Filename already specified: The same filename was specified twice in the same **INPUT** command.

243 Symbol name already specified: The same symbol name was specified more than once in an **XDEF** command.

244 Illegal entry point, command line, or page size value: The values specified in a **COMLINE**, **ENTRY** or **PAGESIZE** command is not even. It must be even.

245 Illegal version/revision number: The version or revision number in an **IDENT** command is not between 0 and 255, inclusive.

246 Description too long: The description in an **IDENT** command is too long. The maximum length for a description is 80 characters.

247 Symbol already exists: The symbol specified in a **DEFINE** command already exists in the ESD table. Symbols may not be redefined with this command.

248 Illegal option or attribute: An illegal (undefined) directive option or task attribute was specified in an **OPTIONS** or **ATTRIBUTES** command.

249 Illegal priority: A priority specified in a **PRIORITIES** command is not a number between 0 and 255, inclusive.

250 Illegal session number: An ASCII-encoded session number (one preceded by a single quote) in a **TASK** or **MONITOR** command is not a number between $0 and $FFFF, inclusive.

251 Buffer length error: A user command line was entered that exceeded 132 characters in length.

252 Buffer overflow: A user command line employing argument substitution greater than 132 characters after substitution of arguments was entered.

253 Argument not found: Argument substitution for an undefined argument was attempted.

Class 3 - Errors in Processing a Relocatable Object File

The general message for errors of this class is:

** ERROR 3nn - Processing relocatable object file - File: <filename>

An error in this class indicates that an error was encountered while processing the relocatable object file named "<filename>". This indicates that the file was not a relocatable object file or it was a damaged relocatable object file. Either way, the linkage editor cannot continue; all class 3 errors are considered fatal and cause an immediate halt to further processing. The errors for this class are:

301 Module(s) not found: A module or modules requested in an INPUT command cannot be found in the specified file.

302 Module already processed: An attempt was made to process a module with the same name as one that already processed from this file.

303 Premature end of file: The end of file in an object module was encountered before the file ended.

306 Illegal relocatable record type: The record type of a relocatable record is not from 1 to 4, inclusive.

307 Error extracting name from ID record: An error was encountered while trying to extract the module name from an ID record.

308 Illegal language type in ID record: The ID record of a module had an illegal language type.

309 No ESD (external symbol definition) records: A relocatable object module does not contain any ESD records.

310 Illegal ESD type: The type of an ESD is not between 0 to 10, inclusive.

311 Error extracting name from ESD: An error was encountered while attempting to extract the name of a symbol or common area from an ESD.

312 Illegal address in ESD: An illegal address was encountered while processing an ESD.

313 Illegal length in ESD: An illegal length was encountered while processing an ESD.

314 ESD record too short: An ESD record is not long enough to contain all the information it should contain.

315 Error extracting address/length in general: A general error was encountered while attempting to extract an address or length from any type of record.

316 Illegal end record: The end record of a module is not of a legal form.

317 Illegal section number in end record: The section number specified in an end record is either not between 0 to 17, inclusive, or the section does not exist.

318 No end of module record: No end record was found at the end of a relocatable object module.

319 Section type conflict: The type of a section is not the same between different relocatable object modules.

320 Section length overflow: A section has become too long. The maximum length of a section is $1000000 bytes.

321 Symbol ESD after non-symbol ESD: A symbol ESD was found after a non-symbol ESD in a module. In every module, all symbol ESDs must appear before the first non-symbol ESD.

Class 4 - Memory Allocation Errors

The general message for this class of errors is:

** ERROR 4nn - Memory allocation conflict

This message indicates that allocation of memory is impossible with the specified input files, sections, segments, etc. An error of this class is fatal and causes an immediate halt to further processing. The types for this class are:

400 Memory conflict: A conflict has occurred that prohibits the allocation of a relocatable section.

401 Out of memory: All memory was allocated but there are still relocatable sections that need allocating.

402 Cannot place absolute section: A conflict has occurred that prohibits the placing of an absolute section where it is required to reside.

403 Section too long: After adding in the lengths of its associated common sections, a section has become larger than the maximum $1000000 bytes.

404 Output file too large: The resulting load module file is too large to fit in available disk space.

405 Maximum address exceeded: Allocating a relocatable section requires logical addresses past the maximum of allowable address ($FFFFFF, $FFFFFFF, or $FFFFFFFF depending on W=24, W=28, W=32).

F

Class 5 - Pass Two Fatal Errors

The general message for errors of this class is:

** ERROR 5nn - Pass two fatal error - File: <filename>

This type of error indicates that an error occurred during pass two processing of the file named "<filename>" and prohibits further processing. This shows that the relocatable object modules in the file needed for input have changed since pass one or there is an error in a relocatable object module that was not detected during pass one, such as a bad data record. All processing stops when this type of error occurs. The types for this error are:

500 ESD index overflow: A relocatable object module being input requires more than 255 ESD indices. One ESD index is required for each relocatable section, absolute section, common section, and external symbol reference. If too many origin statements created this error, the program should be reorganized to take advantage of the 16 available sections.

501 Error calculating entry point address: An error has occurred while attempting to calculate the beginning execution address of the resulting load module.

502 Error calculating command line address: An error has occurred while attempting to calculate the address of where to store the invoking command line.

503 Illegal common name: A non-existent common section name was encountered in a relocatable object module.

504 Illegal section number: A non-existent section number was encountered in a relocatable object module.

505 Illegal symbol name: A non-existent symbol name was encountered in a relocatable object module.

506 Illegal command line ESD: An error was encountered in processing a command line ESD.

507 Data record too short: A data record does not contain enough data.

508 Data record too long: A data record contains too much data.

509 Illegal data record ESDID: The ESDID in a data record that indicates where the data from that record is to go refers to a non-existent ESDID for that module.

511 Illegal ESDID within relocation data: An ESDID within relocation data refers to a non-existent ESDID for that module.

512 Illegal offset size: The flag of a set of relocation data indicates that the size of the offset is not between 0 to 4 bytes, inclusive.

513 Module not found: A module processed in pass one was not found in pass two.

514 Absolute section not found: During pass two, an ESD for an absolute section was encountered for which no ESD was encountered in pass one.

515 ESD index overflow: This error occurs during pass two when creating a relocatable object module if the module requires more than 255 ESD indices. Each relocatable section, absolute section, common section, and external symbol reference requires one index.

Class 6 - Individual Error Messages

There is no general error message for this class but instead, each type of error has its own individual message. All class 6 errors are fatal errors and stop further processing. The individual messages are:

** ERROR 600 - Unresolved references

This error occurs at the end of pass one and shows that unresolved external references still exist, which makes further processing impossible. A list of the unresolved references precedes the error message.

** ERROR 601 - No input files specified

This error, at the end of pass one, shows that the user did not name any input files on the invoking command line or in any user commands. Because of this, the linkage editor has nothing to process and aborts.

** ERROR 602 - Fatal input error

This error indicates that a fatal I/O error has occurred during input.

** ERROR 603 - Fatal output error

This error indicates that a fatal I/O error has occurred during output.

Class 8 - Internal Errors

The general message for errors of this class is:

```
** ERROR 8nn - Internal error
```

A message of this type shows that an error has occurred internal to the linkage editor. Types of error within this class are:

800 This message type indicates that while attempting to assign a value to an external symbol defined within a relocatable section during the listing phase, the linker determined that the section did not exist. This usually indicates that the user defined external symbol(s) within a section of length zero (i.e., a section containing only equates).

801 A message of this type indicates that in an attempt to assign a value to an external symbol defined within a relocatable section during memory allocation, the linker determined that the section did not exist. This usually indicates that the user defined external symbol(s) within a section of length zero.

WARNING MESSAGES

F

Warning messages indicate non-fatal messages that are recoverable. Thus, when one of these types of errors occurs, a warning message is generated and processing continues normally. The general form is:

```
** WARNING 7nn - description
```

where:

7nn	is a three-digit warning number starting with "7".
description	is a description of the error that occurred.

The warning messages are:

```
** WARNING 700 - Undefined symbol:  <symbol>
```

This warning indicates that an **XDEF** command specified a symbol "<symbol>", that is not in the current table of **XDEF**ed symbols. In other words, it has not appeared as an **XDEF** in any of the relocatable object modules processed so far. The processing of the **XDEF** command proceeds as if the offending symbol "<symbol>" were not in the command.

** WARNING 701 - Multiply defined symbol: <symbol name>

This warning indicates that the symbol "<symbol name>" is multiply defined. This means that the symbol was **XDEF**ed in more than one relocatable object module, i.e., defined in more than one place. The action taken will be to use only the first occurrence of "<symbol name>" as its defining occurrence and to ignore all further definitions.

** WARNING 702 - No END command, assumed

This warning indicates that the end of file in the user commands from a file was found before an **END** command was encountered. The action taken is to manufacture a fake **END** command and proceed.

** WARNING 703 - Section not assigned, section not loaded: nn

This warning shows that during the processing of a relocatable object module in pass one, a section definition was found for section number "nn" which was not assigned to a segment. The section definition is ignored and processing continues.

** WARNING 704 - Conflicting XREFs: <symbol name>

This message indicates that two **XDEF**s to the same symbol, "<symbol name>", conflict because they require the symbol to be defined in different sections. The action taken is to allow the symbol "<symbol name>" to be defined in any section and processing continues.

** WARNING 705 - Relocated value too large, value truncated: at $<address>

This message indicates that while processing relocation data for data to be put at hex address "<address>", the resulting value was too large to fit into the number of words set aside. The action taken is to truncate enough from the high order so that the result will fit. For example, a jump to subroutine where the address to jump is too far away.

** WARNING 706 - Section not assigned, symbol not loaded: <symbol name>

This message shows that an **XDEF** was encountered for the symbol "<symbol name>", while processing a relocatable object module in pass one. However, the section defining the symbol has not been assigned to a segment. The **XDEF** is ignored and processing continues.

** WARNING 707 - Module appears more than once: <module name> in <filename>

This message indicates more than one module was encountered with the same name ("<module name>"), while processing the file named "<filename>" in pass two. The first module encountered is processed while later modules with the same name are ignored.

This could occur if a library file contained two modules with the same name and a search of the file during pass one indicated that the second module was needed. However, in pass two the first module with the desired name will be processed, and when the second module with the same name is encountered, this warning message is generated. This could be serious since the data in the first module is processed in pass two according to the information acquired from the second module in pass one. Obviously, every module in a file should have a unique name.

** WARNING 709 - Unable to include in debug file <filename>

This message indicates that the linker attempted to append an .RS file to its .DB file, which it could not find.

** WARNING 710 - Attempt to locate short section <section> at <address>.

This message indicates that the user attempted to locate a short relocatable section "<section>" in an address range not completely contained within the first 32Kb or the last 32Kb of addressable memory. The starting address of the section is given by <address>.

APPENDIX G

MVME12x-SPECIFIC INFORMATION

Special linking procedures must be followed when linking tasks to run on an MVME12x system. When loading programs on these systems an error message, "SEGMENT ERROR-TASK LINKED INCORRECTLY", may appear for tasks that have run unaltered on other systems.

Special care must be taken to run NON-position independent programs on an MVME12x system. Due to the nature of the instruction cache, segments must appear on even 1K boundaries to run properly. The following guidelines should be followed.

a. A position-independent program must be linked with the **ATTRIBUTES "P"** option specified in the link input commands. The loader makes the proper adjustments to align the task segment boundaries. Using the **DUMP** utility, display sector 0 of the load module to see if the task was linked with this attribute. If bit 3 of byte $14 is set, the task was linked with the **ATTRIBUTES "P"** option. This does not guarantee, however, that the task is position independent. It simply tells the loader to treat the task as if it were. This allows for movement of the segments relative to one another as they are loaded into memory.

b. If the task is NOT position independent, it will be necessary to relink the task with the **PAGESIZE** interactive command. The start address for each segment must be on 1K boundaries. For example, assume a task is being linked to run on the MVME121 and the input commands are:

```
=LINK ,TASK,TASK;HAMIXS
M68000 Linkage Editor Version x.xx
Copyrighted 1985 by Motorola, Inc.

>SEG SEG0:5,3   $0000
>SEG SEG1(RL):0-2
>SEG SEG2:4
>INPUT MAIN
>INPUT SUBRTN
>INPUT A,B,C
>LIB  MYLIB
>END
```

The start and end addresses of the segments are:

	START	END	LENGTH
SEG0:5,3	$0000	$06FF	$0700
SEG1(RL):0-2	$0700	$22FF	$1B00
SEG2:4	$23FF	$25FF	$0200

Note that only the first segment starts on a 1K boundary. To force all the segments to start on a 1K boundary, use the **PAGESIZE** command:

```
=LINK ,TASK,TASK;HAMIXS
M68000 Linkage Editor Version x.xx
Copyrighted 1985 by Motorola, Inc.

>PAGESIZE     1024
>SEG SEG0:5,3   $0000
>SEG SEG1(RL):0-2
>SEG SEG2:4
>INPUT MAIN
>INPUT SUBRTN
>INPUT A,B,C
>LIB   MYLIB
>END
```

The interactive command **PAGESIZE 1024** forces the segments to appear at addresses:

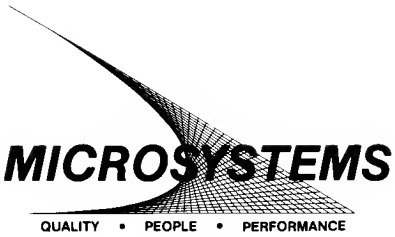
	START	END	LENGTH
SEG0:5,3	$0000	$06FF	$0700
SEG1(RL):0-2	$0800	$22FF	$1B00
SEG2:4	$3000	$25FF	$0200

The segments are now forced to begin on even 1K boundaries whether the MMU is enabled or not. If running without an MMU, link a NON-position independent task to run at the desired physical address. The loader attempts to load the task at the specified location if possible. If no memory is available at the linked location, a load error occurs.

G

INDEX

SUGGESTION/PROBLEM REPORT



Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microsystems
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: _____ Manual: _____

COMMENTS: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please Print

Name _____ Title _____

Company _____ Division _____

Street _____ Mail Drop _____ Phone _____

City _____ State _____ Zip _____

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Four Phase/Motorola Customer Support, Tempe Operations
(800) 528-1908
(602) 438-3100

MOTOROLA

MOTOROLA ***Semiconductor Products Inc.***

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

17630 PRINTED IN USA (1/86) MESSENGER 3500